

AD-A031 481

AUERBACH ASSOCIATES INC PHILADELPHIA PA
A DEDUCTIVE SYSTEM FOR INTELLIGENCE ANALYSIS.(U)
JUL 76 I.L GOLDHIRSH, R CARSON

F/G 9/2

UNCLASSIFIED

RADC-TR-76-199

F30602-74-C-0250
NL

1 OF 2
ADA031481



AD A031481

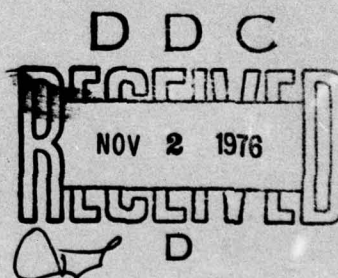
RADC-TR-76-199
Final Technical Report
July 1976

A DEDUCTIVE SYSTEM FOR INTELLIGENCE ANALYSIS

Auerbach Associates Inc.

Approved for public release;
distribution unlimited.

ROME AIR DEVELOPMENT CENTER
AIR FORCE SYSTEMS COMMAND
GRIFFISS AIR FORCE BASE, NEW YORK 13441



This report has been reviewed by the RADC Information Office (OI) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public including foreign nations.

This report has been reviewed and is approved for publication.

APPROVED:

Robert N. Ruberti

ROBERT N. RUBERTI
Project Engineer

APPROVED:

Howard Davis

HOWARD DAVIS
Technical Director
Intelligence & Reconnaissance Division

FOR THE COMMANDER:

John P. Huss

JOHN P. HUSS
Acting Chief, Plans Office

Do not return this copy. Retain or destroy.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER RADC-TR-76-199	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) A DEDUCTIVE SYSTEM FOR INTELLIGENCE ANALYSIS.	5. TYPE OF REPORT & PERIOD COVERED Final Technical Report June 1974 - April 1976	
7. AUTHOR(s) Isadore L. Goldhirsh Richard Carson	6. PERFORMING ORG. REPORT NUMBER N/A	
9. PERFORMING ORGANIZATION NAME AND ADDRESS Auerbach Associates Inc 121 North Broad Street Philadelphia PA 19107	8. CONTRACT OR GRANT NUMBER(s) F30602-74-C-0250	
11. CONTROLLING OFFICE NAME AND ADDRESS Rome Air Development Center (IRDT) Griffiss AFB NY 13441	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 31025F IDHS0434	
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) Same	12. REPORT DATE July 1976	
	13. NUMBER OF PAGES 116	
	15. SECURITY CLASS. (of this report) UNCLASSIFIED	
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A	
16. DISTRIBUTION STATEMENT (of this Report) Approved for public release; distribution unlimited.		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report) Same		
18. SUPPLEMENTARY NOTES RADC Project Engineer: Robert N. Ruberti (IRDT)		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) Deductive Processing Intelligence Data Handling Artificial Intelligence		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) This report describes concepts and a design architecture of an inference system in a fact-base supported information system. The architecture is specialized to efficiently handle the separate functions of deduction development, fact search, deduction verification (by fact base retrieval and or inferred information) and user dialogue. Distinctive features of the design include: (1) deduction development strategies based on performance success histories,		

DD FORM 1 JAN 73 1473

EDITION OF 1 NOV 65 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

391 061

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

- (2) operation of deduction instantiation (by real or inferred facts) on a scheduled basis,
- (3) special inference system management of acquired facts including determination of fact retrieval requests that minimize search time and retrieval volume, and
- (4) special analyses that curtail non-useful deduction development.

A program design is presented demonstrating separation of function (modularity), and multi-user real-time capability.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

PREFACE

The work covered in this report was accomplished under contract F30602-74-C-0250 to develop the concepts and a design of an inference system appropriate to an information system within an intelligence environment. The report was prepared by Isadore L. Goldhirsh and Richard Carson of the staff of AUERBACH Associates, Inc. Grateful acknowledgement for special consultation is given to Dr. J. Minker of the University of Maryland, to Dr. Richard LeFaivre of Rutgers University, and E. Scherneck of the Auerbach staff. The work of this report was developed upon the foundation set in a prior report entitled "Design Concept for an Augmented Relational Intelligence Analysis System (ARIAS)", August 1973, by Dr. J. Sable, et al. Dr. J. Sable was also the program manager for this report and provided valuable critical and technical assistance. The support of Rome Air Development Center and Mr. Robert Ruberti, the Technical Monitor is gratefully acknowledged.

ACCESSION for	
NTIS	White Section <input checked="" type="checkbox"/>
DGC	Defi Section <input type="checkbox"/>
UNANNOUNCED	<input type="checkbox"/>
JUSTIFICATION	
BY	
DISTRIBUTION/AVAILABILITY CODES	
Dist.	AVAIL. and/or SPECIAL
A	

D D C
RECEIVED
NOV 2 1976
D

EVALUATION

Design specifications for a deductive inference system have been delivered under Contract F30602-74-C-0250. When implemented as an adjunct to an intelligence information system, an inferential system will improve analyst-data base interactive capabilities, i.e. , augmenting information retrieval, reducing data storage requirements, providing interactive hypothesis testing. Future application of inferential data analysis is planned for the Scientific and Technical Information System (STIS) at the Foreign Technology Division. This development is included as part of TPO #4, Intelligence Data Handling.

Robert N. Ruberti
ROBERT N. RUBERTI
Project Engineer

TABLE OF CONTENTS

<u>P</u> ARAGRAPH	<u>T</u> ITLE	<u>P</u> AGE
<u>SECTION I. INTRODUCTION</u>		
<u>SECTION II. INFERENCE SYSTEM CONCEPTS AND ARCHITECTURE</u>		
2.1	PRELIMINARY DISCUSSION OF RELATIONS, LITERALS, DEDUCTIONS, AND QUERIES	2-7
2.1.1	The Literal	2-7
2.1.2	Types of Literals	2-7
2.1.2.1	Type of Variable Domains	2-8
2.1.2.2	Range of Variable Domains	2-8
2.1.3	Types of Deductions	2-10
2.1.4	Answers to Queries	2-11
2.1.5	Interpretation of Formal Implication in an Information System	2-11
2.2	DEDUCTION	2-12
2.2.1	Unification	2-12
2.2.2	Resolution	2-13
2.2.3	Deduction Nets, Trees and Linear Chains	2-13
2.3	DEDUCTION MECHANISM	2-14
2.3.1	Ordered Linear OL-Deduction Method	2-14
2.3.2	OL-Deduction Compatible with Set of Support Strategy	2-17
2.3.3	Selective Negative Linear Resolution (SNL).....	2-18
2.3.4	Paramodulation	2-18
2.3.5	Deduction Simplification and Elimination: Post Processing Operations	2-19
2.4	FACT DETERMINATION	2-21
2.4.1	Summary of Sources of Facts for Deduction Evaluation	2-22
2.4.2	Deduction and Verification: A Two Stage Procedure..	2-22
2.4.3	Effect of Actual vs. Statistical Fact Data on Deduction Development	2-25
<u>SECTION III. DEDUCTION DEVELOPMENT STRATEGY</u>		
3.1	CONTEXT SETS	3-1
3.1.1	Context Set Formation	3-2
3.1.2	Context Set Hierarchy	3-3
3.1.3	Context Set Weights and Bridges	3-3
3.1.4	Application of Context Sets	3-4

TABLE OF CONTENTS (CONTINUED)

<u>PARAGRAPH</u>	<u>TITLE</u>	<u>PAGE</u>
3.1.5	Deriving Context Weights for Literals	3-5
3.2	DEDUCTION ORDERING AND SELECTION FOR DEDUCTION DEVELOPMENT	3-5
3.2.1	Literal Ordering	3-5
3.2.1.1	Least Likelihood of Refutation Criterion for Literal Ordering in a Deduction	3-7
3.2.1.2	Easiest to Refute Criterion for Literal Ordering in a Deduction	3-8
3.2.1.2.1	State 1, Deduction Development: Literal Ordering Criteria	3-8
3.2.1.2.2	State 2, Fact Determination: Ordering for Fact Retrieval and Intersection	3-9
3.2.2	Deduction Selection	3-12
3.2.3	Rule Selection for Deduction Development	3-13
3.3	DEDUCTION PRUNING	3-14
 <u>SECTION IV. PROGRAM DESIGN</u> 		
4.1	PROGRAM MODULES AND INFORMATION FLOW	4-1
4.1.1	Query Module	4-1
4.1.2	Deduction Tree Management Module (DTMM)	4-5
4.1.3	Deduction Development Module (DDM)	4-5
4.1.4	Deduction Resolution Module	4-8
4.1.5	Fact Search Module	4-8
4.1.6	Fact Generation Algorithms	4-11
4.1.7	Other Modules	4-11
4.2	DEDUCTION PROGRAM DATA DICTIONARY	4-11
4.2.1	Deduction (Tree) Table	4-11
4.2.2	System Rules Table	4-14
4.2.3	Deduction Development Order Table	4-14
4.2.4	System Literal Table	4-14
4.2.5	Deduction Merit Ranking Table	4-16
4.2.6	Deduction Fact File Base (DFFB)	4-16
4.2.7	Performance Information Table (Statistics)	4-17
4.3	SOME PROGRAMMING DESIGN TECHNIQUES	4-17
4.3.1	Resolution Unification Mechanism	4-17
4.3.2	Factoring Mechanization	4-18

SECTION I. INTRODUCTION

"An inference system derives conclusions (inferences) from facts and premises." This Webster Dictionary definition is interpreted to include inferences that are new facts, not heretofore assembled, and existing facts available within a body of preassembled fact data. Much of inferential logic in current literature deals with former, that is the generation of new facts implied by a body of fact data and a set of premises. This is evident in the areas of problem solving and theorem proving. However, in the arena of information systems, emphasis is upon the latter, that is, upon finding facts that currently exist in an assembled body of fact data, or that may be generated upon request by specialized fact generators (computer programs). The facts sought are those that can satisfy an information query. The query may be a primitive direct request as "have you any X, Y and Z facts?"; or the query may be as complex as "what facts have you that satisfy... (and a list of interrelated conditions are stated)?". The query may also be posed with conditions on the number of answers desired and with qualifications on the answers (e.g., precision, credibility).

If the fact data of an information system were organized to be explicitly responsive to every valid query, the logical (data relational) organization

and the efficiency of fact base storage, in regard to redundancy needs and accessibility, would likely acquire a superstructure, which might be overwhelming even for a relatively small information system. As a practical matter, to circumscribe such overhead, an information system is supplemented by rules that reflect data relationships among facts of the fact base, so that not all information need be explicit in the fact base. The rules may also provide representation of fact relationships that are alternates to relationships explicit in the fact base, to afford alternative, less costly, accesses to facts.

SECTION II. INFERENCE SYSTEM CONCEPTS AND ARCHITECTURE

An inference system is a subsystem within an information system. It lies between the information system Dialogue package and Fact Base, with formal interfaces as shown in Figure 2-1. The primary components of the inference system are:

- (1) Query Analyzer
- (2) Deduction Mechanism
- (3) Deduction Rules
- (4) Fact Search Mechanism
- (5) Answer Responder.

A Performance History Analyzer component is included as a sixth component to support the inference system in a learning or adaptive mode so that it may cope with changing operational situations.

The Deduction Rules (or just Rules) component contains (logical relational descriptions (expressed in this report by the Predicate Calculus) that:

- (1) characterize the fact base and its inter-relationships,
- (2) generate special data sets, and
- (3) contain assertions, whose fact satisfiability is not necessarily assured.

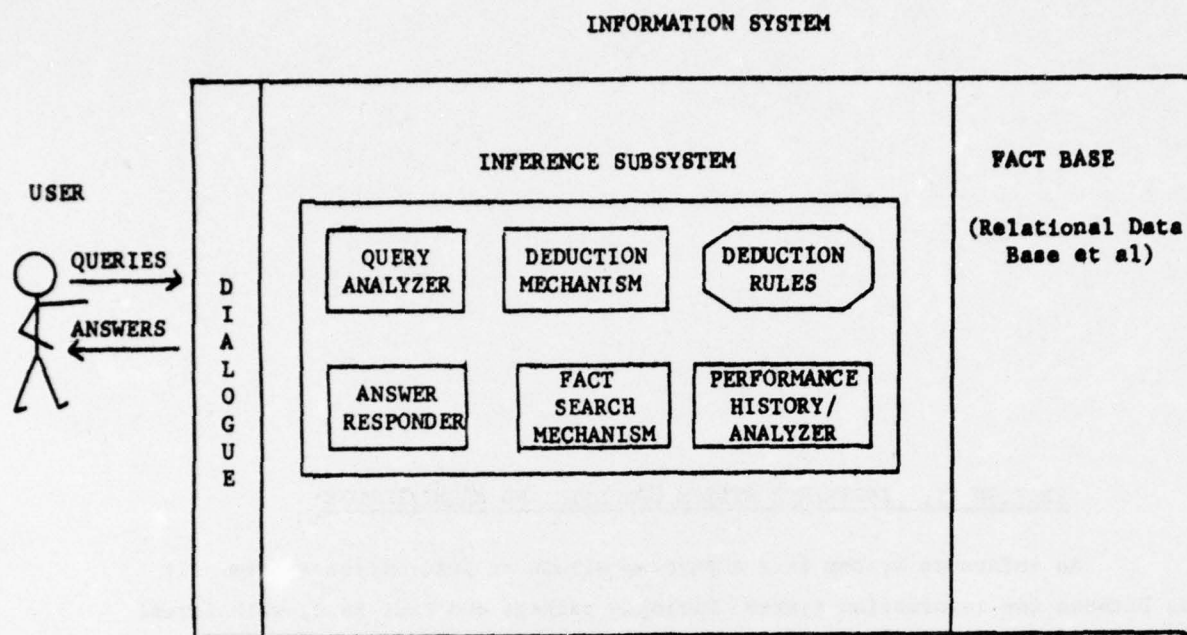


Figure 2-1. Inference Subsystem in an Information System

The Rules component is extendible to include rules that translate and enable alternate versions of a query. This permits the same requirement to be expressed differently by users, and may properly be regarded as providing a query language translation capability within the inference system.

The inference system operates as follows. A user query is passed to the inference system by the information systems Dialogue. The query is then processed by the Query Analyzer, which prepares it in the proper logical form for deduction processing. The Deductive Mechanisms of the inference system attempt to ascertain the satisfiability of the query by facts existing within the information system fact base, by facts generatable by special generators, or by some of the Deduction Rules.

The Deduction Mechanism generates deductions by employing logical inference techniques, that are well established in the literature; see References, in Appendix F. The different approaches cited in the literature are generally equivalent in that they derive the same results, utilizing the logic of the predicate calculus. There are variations among approaches designed to simplify operations for computer computation; and we have chosen and adapted, with this purpose, the OL-Deduction approach described by Chang and Lee, Reference 1.

Given a workable deduction approach, a principle concern has been to devise a strategy that helps to narrow the almost limitless range of possible deductions that may be generated in the search for useful, potentially satisfiable deductions. On this matter, there is little of substance in the current literature, and very little in the way of practical experience, see Reference 6. Four approaches, however, are identified and to various extents practices. They are:

- (1) Deduction Expression Characteristics. Presume success probabilities (e.g., likelihood of a deduction being satisfied or leading to another satisfiable deduction) by correlating success probability values with certain aspects of the deduction statement, such as, the number of logical expressions in the statement. Chang and Lee, for example, enumerate 14 such content-independent characteristics of a deduction statement.

This approach leaves much to be desired since it presumes, in effect, that the satisfiability of a deduction correlates with the expression of a deduction rather than with the "meaning" of the deduction in the context of an information system. It suggests, for example, that short deduction statements are more likely to be satisfied than long deduction statements.

- (2) Breadth versus Depth. Most deductive systems generate deduction trees in which each node of the tree is a potential deduction resolved from its parent nodes on the tree. The deduction tree can be encouraged to expand depth-wise or breadth-wise, arbitrarily. Contentions arise as to the merit of developing a tree along its breadth prior to descending in tree depth, visa versa, or the myriad of in between. Deduction development approaches based on breadth/depth considerations are even more limited than (1) above in regard to reflecting the content of deductions, and tend to wander into meaninglessness as the deduction tree expands.
- (3) Success History. This a probabilistic approach in which the likelihood (probability) of a successful (i.e., satisfiable) deduction or of leading to a successful deduction is estimated on the basis of a deduction's success history. The approach is, in this sense, content related. This approach collects statistical performance data on logical expressions appearing in deductions, to compute probabilities. The approach is presumed to work well in a stable information system, that is, in one whose fact base and rules change slowly in comparison with usage.
- (4) User Interactions. The user (inquirer) is engaged in this approach in directing deduction development. An exchange is provided via the Dialogue of the information system. For practical operation, this approach does not necessarily involve the user in evaluating each potential deduction (since thousands of potential deductions could be generated for difficult queries), rather, milestones or conditions are set as criteria for engaging the users aid. Considerations relating to these are:
 - (a) shall a high probability deduction be pursued which has a predicted heavy computation load compared to other potential deductions?
 - (b) shall the deduction process be pursued further, having exceeded a prescribed computational budget allotment?

- (c) will the user wish to restate his query based on partial results thus far achieved?

This approach is highly effective in that it brings the user and the information system together, in a dialogue, for answering queries. It does not, however, absolve the inference system from doing its part to automate the query answering service.

In this report we have placed our emphasis for the design of deduction development upon the content related approaches, namely, the Success History and User Interaction approaches, putting aside as incidental the other two.

At the outset, we had recognized that the heaviest computation burden (use of resources) and processing delays lay not in the Deduction Mechanism, but rather in the Fact Search Mechanism, (refer to Figure 2-1). Whereas, the Deduction Mechanism might generate a large quantity of deductions, these could be computed very rapidly by high-speed computer; and, assuming low cost bulk storage availability and an overlay storage capability, a low cost could be expected for computer storage and computing resources by the deduction process. On the other hand, the Fact Search Mechanism, which generates specific facts and collects selected fact data from the information system fact base, entails extensive and expensive data retrieval and data manipulation operations. Fact retrieval requests transmitted by the Fact Search Mechanism to the information systems fact base initiate large data search and retrieval operations in that area of the information system, along with substantial delays associated with the required searches through the fact base. For these reasons, great attention was given to the Fact Search Mechanism of the inference system, to its assigned functions and to its design.

The Fact Search Mechanism (FSM) receives, from the Deduction Mechanism, potential deductions or parts of deductions which may be satisfiable by facts contained within, or generatable by, the information system. The FSM acquires the desired facts from the information system, via formal communications. Retrieval requests are optimally organized and issued by the FSM to the information system in a manner to minimize overall search and communications load. Fact data received by the FSM are filtered of non-satisfying facts and the results are assembled for query answering.

It should be noted that, in much of the literature by both retrieval from the fact base and through deductive generation are procedurally intermixed. There are two practical reasons why this is a generally unworkable procedure.

- (1) Fact data derived from the fact base or from special fact generators are not immediately available for deduction verification, requiring normally a retrieval time interval. Such delays hold up deduction development. It is generally better to continue deduction development (to a point!) without waiting for fact verification, even though this risks generating some non-useful deductions.
- (2) Determining facts within the deduction procedure, by generating a collection of deductions, each of which is a derived fact, is inefficient (expands the number of deductions and complicates the fact determination procedure) compared to utilizing specialized fact generators (computer programs) or compared to retrieving the collection of facts, in batch mode, from the Fact Base.

While we have not excluded from the Deductive Mechanism the capability of performing individual fact generation, we have designed the FSM to strongly support fact determination and verification, on a batch basis, with facts derived from the information systems Fact Base and from specialized fact generators.

A complement component to the Query Analyzer of the inference system (see Figure 2-1) is the Answers Responder. This component has two functions:

- (1) to prepare dialogues with users and convey results, via the information system Dialogue,
- (2) to convey information to the Deduction Mechanism concerning the satisfiability of deductions.

With regard to the latter, the sooner the Deductive Mechanism can learn whether or not a deduction is satisfiable, the better it can guide deduction development.

The final major component of the inference system is the Performance History Analyzer. This component collects success-failure data, processing resource costs, and performance speed, and generates periodic probability data, needed to guide the Deduction Mechanism in its deduction development.

2.1 PRELIMINARY DISCUSSION OF RELATIONS, LITERALS, DEDUCTIONS, AND QUERIES

2.1.1 The Literal

The smallest logical unit in the Inference System is the simple relational statement, sometimes called an atom. A relational statement has the form $R(a_1, a_2, \dots, a_n)$. We may say that the symbol R stands for an n -ary relation when it takes n domain elements or terms, a_1, \dots, a_n . For example in the relational statement $\text{Location}(\text{Smith}, \text{Phila})$, Location expresses a binary relation between the terms Smith and Phila . A literal L_i is either a relational statement (atom) or its negation, and hence we speak of positive or negative literals. If we have $L_1 = R(a,b)$ and $L_2 = \bar{R}(a,b)$, then L_1 and L_2 are complementary literals. It is customary to refer to L_1 as a positive and L_2 (or \bar{L}_1) as a negative literal.

In the inference system all logical statements (propositions, rules, queries, etc.) will be put into a disjunctive normal form so that the scope of each negation is a single relational statement (atom). Therefore, we can treat all statements as a disjunction of literals. We call a disjunction of literals a clause and customarily omit the logical operators in clauses so that a statement such as

$$A(x,y) \wedge B(y,z) \Rightarrow A(x,z)$$

will appear as

$$\bar{A}xy \bar{B}yz Axz$$

in clause form. This stands for the disjunctive normal form $\bar{A}(x,y) \vee \bar{B}(y,z) \vee A(x,z)$. We further assume that all term variables in the clause are universally quantified, that is, true for all values of x, y , and z , unless otherwise delimited by the deductive system

2.1.2 Types of Literals

There are several types of literals in the inference system. One type identifies classes of facts in the fact base, and are denoted as fact literals. These literals may be passed directly to the fact base for immediate

fact retrieval without further logical analysis of the literal. A second type of literal is used primarily to inter-relate other literals, and, therefore, to construct inference rules and queries. This type of literal may also be used to infer facts or classes of facts that are neither available in the fact base nor provided by the fact generators. These inferred facts are not verified by the information system, and if accepted, are accepted on faith. This is discussed further in Section 2.1.3.

Rule, query, and deduction clauses may contain one or both literal types. A rule containing only fact literals provides a means for defining alternative literals for fact retrievals - an important feature of an inference system concerned with optimal fact retrieval performance. The set of fact literals is recognized and maintained by the inference system as "one-literal" rules. Also maintained with each fact literal are certain statistical data reflecting the fact domain of the literal, the cost of fact retrieval for the literal, and other information useful in guiding deduction development; see section 3, Deduction Development Strategy.

2.1.2.1 Type of Variable Domains

The variables in a clause are all formally universally quantified, that is, the proposition expressed by the clause is to be satisfied for all values of the variables. However, the type of entities, objects, etc. in the domain of a given relation which make it semantically meaningful is often quite restricted, and may be a small subset of all entities in the system. The Inference System will use this information to restrict the number of tests which must be made to verify the validity of a proposition. For example, in the rule $P(X,Y) \wedge P(X,Z) \Rightarrow M(Y,Z)$ (where P means parent and M means mate), people, animals and (some) plants may be substituted for the X, Y and Z terms. The meanings (types of allowable substitutes) for terms of each relation is recognized and controlled by the inference system.

2.1.2.2 Range of Variable Domains

In classic logic, the use of the quantifier all means that the variables are unrestricted. In a real information system, the meaning of the

quantifier all is restricted to those entities available within the information system. In a practical way, "all" is further restricted to those entities which are available within a budget of time and resources. A working information system must specify (or estimate) the domain of each term in a relation. In particular it must specify (or estimate) the number of entities in that domain. We call this quantity the range of the domain. For example, in "all parents of people", the range of the domain of parents to a single child could be specified as two. The number of parents may be limited to a current count of entries in a parent fact file, or a practical range limit of, say, 10,000 may be set.

In a working information system, facts potentially satisfying queries take time to derive, and may come forward irregularly and in disorder, as appropriate fact banks are searched. If the query is satisfied prior to retrieving all facts, search may be terminated before being fully completed. This is illustrated by the following query:

Query: Find two workers in factory F that have skill in electronics.

$(\exists x) (\text{Works-at } (x, F) \wedge \text{Skill } (x, \text{Electronics}))$

Range (x) = 2

Type (x) = people

Procedure: We might proceed to first find and generate a file of all workers Y in factory F, and then to select from this file a subfile of all workers with electronics skills; and then to select the first two of this subfile for the answer to our query. We might instead, however, search the factory file one at a time, testing for an entry that has an electronics skill, and continuing until two are found.

It should be noted that classical logic tends to express itself by the first of these two procedures, taking advantage of the elegance of expression provided by the "all" concept. The second procedure may, on the other hand, be more economical involving less search activity. In practice, the best method will sometimes be one or the other, depending on factors of fact logical organization and physical storage, the performance of fact retrieval

mechanization, interdependency of literals of a query and the number of facts requested by the query in relation to the number available within the information system.

2.1.3 Types of Deductions

A deduction is a logical statement generated by the inference system, starting, for example, with a user query and a set of rules. There are two general types of deductions. Each of these is characterized by the extent to which the information system can supply information, that is real facts, that satisfy the deduction.

- (1) Verified deductions. These are deductions for which the information system supplies information, that is, concrete facts that support the deduction. The user is supplied these facts and the deduction, as a response to his query. It is clear that in this type of deduction, all literals of the deduction are of the fact literal type.
- (2) Inferred deductions. These are deductions for which the inference system does not supply facts from the fact base or from fact generators, to support the deduction. The deduction is taken on faith. This type of deduction occurs, for example, when a set of rules that are applied to a query are accepted as representing "true" relations. As an illustration, an inference might be that "every paid employee has performed a measurable service"; a verified deduction would require examining the accounting performance summary files of paid workers in the information system.

Inferred deductions are called inferences, and divide into two sub-types: Those inferences that are verifiable by facts from the information system, but have not been verified; and those that are not verifiable within the means of the information system. This division is useful because a user recognizing the costs and delays associated with fact verification may feel satisfied with partially verified results.

We note that the literals of non-verifiable inferred deductions include at least one non-fact literal. The literals of verifiable inferred deductions consist of fact literals only.

2.1.4 Answers to Queries

The inference system generates deductions as a means of answering queries. Deductions may be classed into acceptable deductions and unacceptable deductions. Unacceptable deductions are deductions which are either not meaningful or are unacceptable to a user as a response to his query. This is quite general and can even apply to verified deductions if the supporting information does not meet "extra" user requirements - which are conveyable to the inference system via a dialogue. Also, an acceptable deduction may not necessarily fully satisfy a query. For example, if a deduction satisfies a query with 10 facts, but at least say 20 facts are desired, it will be necessary to continue the development of deductions in the information system until the desired quantity of facts is derived - or the search is aborted.

2.1.5 Interpretation of Formal Implication in an Information System

In our information system every logical statement is expressed as a clause; that is, as a disjunction of literals. The clause may be equivalent to an implication; for example, the clause

$$\bar{A}(x) \vee \bar{B}(x,y) \vee C(x,y)$$

is the same as the implication

$$A(x) \wedge B(x,y) \Rightarrow C(x,y).$$

The interpretation of the clause (and therefore of the implication) in our information system is "find all values of x and y which satisfy the clause". The result will be n vectors:

$$S_i = (X_i, Y_i), i = 1, 2, \dots, n.$$

where the X_i and Y_i are constants (entities) in the domains of variables x and y respectively.

Each vector S_i provides a single answer that satisfies the clause, and its equivalent implication statement.

In regard to the implication statement, this represents a more general interpretation than the classic "true or false" interpretation, that makes no distinction as to possible answers within subdomains. Re-stating this difference, we have in the classic logic that, assuming universally quantified variables, $A(x) \wedge B(x,y) \Rightarrow C(x,y)$ is a true statement only if it holds for all x and y . In the information system, $A(x) \wedge B(x,y) \Rightarrow C(x,y)$ is a statement to be verified as true, by vectors (X_i, Y_i) that can be discovered within the information base of the system to satisfy the implication. It may be curious to some logicians, that the null set may be, for some cases, the only discovered answer to a universally quantified statement. More likely, some non-null answers will be found. The classic conception of implication can be highly effective in an abstract universe where immutable laws govern the interrelationships of all objects. It can be appreciated, however, that an information system is not an abstract universe. In an information system immutable laws governing information are scarce and information is acquired by examinations of a multitude of non-universal rules and by an abundance of detailed fact searching. The classic implication statement is not sufficient, therefore, for a "real" information system and must give way to the more effective general implication statement interpretation presented here.

2.2 DEDUCTION

2.2.1 Unification

Within the Deductive System, all logical statements in the predicate calculus, e.g., queries, rules and deductions, are represented by "clauses". Symbolically, a clause C designates a disjunction of literals $\{L\}$ by, $C = L_1 \vee L_2 \dots \vee L_n$, where $L_i \vee L_j$ means either L_i or L_j or both. The literals themselves are expressions involving "terms". Terms need not be logically defined to the deduction mechanism beyond their uniqueness; their meaning, however, is that they specify information that together with relational clauses, result in statements of fact. Thus, for $L(X)$, X are all terms satisfying the assertion $L(X)$. A literal $L(X)$ is not the same as a literal $L(Y)$, when X and Y refer to different sets of terms (entities). When X and Y are not disjoint, a process called "unification" can be applied. This involves a procedure of substitution and occurs a great deal in deduction mechanisms because the terms of literals in

the rules of an inference system tend to have wide domains. When the domain of a term is universal, that is, it covers all facts, then the term may be called a universal variable. Generally, this is not the case and the domain of each (variable) term must be specified. For example, in $L_1(X,Y)$ and $L_2(U,W)$, if $Y \subset U$, we can identify $L_1(X,Y)$ and $L_2(Y,W)$ as two literals that have been unified. This unification procedure is extendible to clauses with common domains, and as we shall show, is appropriate to "resolution", a procedure for generating deductions from two parent clauses.

2.2.2 Resolution

Deductions are generated in our inference system by a method called "Resolution", illustrated by the following:

Given two clauses $L_1(X) \vee L_2(Y)$ and $\bar{L}_1(Z) \vee \bar{L}_3(Z)$, if Z is a variable term whose domain includes X , we may let Z take the values of X . We can then "resolve" these two clauses to form a deduction (resolvent) by

$$[L_1(X) \vee L_2(Y)] \wedge [\bar{L}_1(X) \vee \bar{L}_3(X)] \Rightarrow [L_2(Y) \vee \bar{L}_3(X)]$$

In discussing resolution, we will limit ourselves to binary resolution, as illustrated in the preceding example, and we will refer to the two starting clauses as parents, and to the deduction clause as the resolvent clause. We also make reference to the positive form of the literal of the parents that does not appear in the resolvent, as the "resolution literal". In the preceding example, $L_1(X)$ is the resolution literal. We will frequently find it useful to retain in a resolvent clause the resolution literals associated with it, and will mark the resolution literals appropriately to distinguish them from other literals in the clause. A deduction that is the result of a chain of resolutions can, therefore, display a number of these marked resolution literals, one for each link in the chain.

2.2.3 Deduction Nets, Trees and Linear Chains

A deduction lattice network can be generated by utilizing starting "clauses" (assertions and rule clauses), followed by resolvent clauses as well as

starting clauses as parents of new resolvent clauses. A (partial)* deduction tree is a deduction net in which at least one of the parents of each resolvent is a non-resolvent clause (sometimes called an input clause). A linear chain deduction is a path of a tree deduction. These are illustrated in Figure 2-2.

2.3 DEDUCTION MECHANISM

We have adopted for our Deduction Mechanism the method of ordered linear deduction (OL-deduction) described by Chang and Lee, Reference 1, Appendix F. The OL-deduction method operates upon a procedure in which the negation of an assertion is attempted to be refuted. When the negation of an assertion is successfully refuted, the assertion is, thereby, verified. The refutation procedure initiates with a set of rules and an assertion presented in negated form for refutation.

2.3.1 Ordered Linear OL-Deduction Method

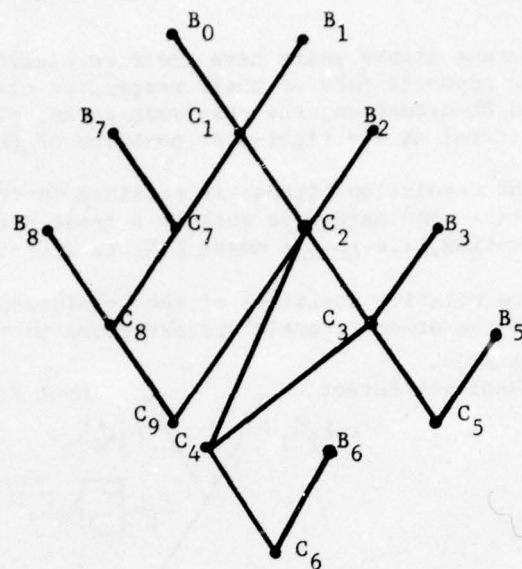
OL-deduction is a deduction tree development procedure containing certain restrictions on the construction of deductions and on the representation of resolvent and resolver deduction clauses. In this report, OL-deduction ordering restrictions are as follows:

- (1) Two parent clauses are used to generate, by the procedure of resolution, a single deduction--resolvent clause. In OL-deduction, a pair of parents may not have a common ancestor, that is, the deduction graph may not have the form of a general lattice. This restriction, in effect, means that each parent pair must consist of one input clause (rule or query clause) and one resolvent clause, except for initial tree deductions. This restriction is compensated by a later operation called "reduction".

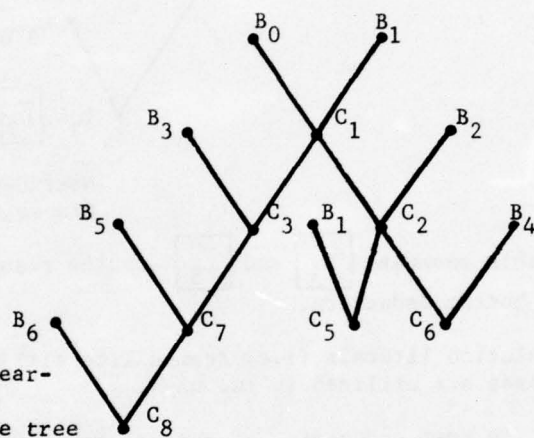
* We shall drop the term "partial" and use the expression "tree" in what follows. The partial tree is really a limited lattice since an input clause may be parent to more than one deduction node of the tree.

Starting Clauses (B_0, B_1, \dots, B_n)

Deduction Lattice



Deduction Tree



Note: Double appearance of B_1 , which actually makes the tree a lattice.

Deduction Linear Chain

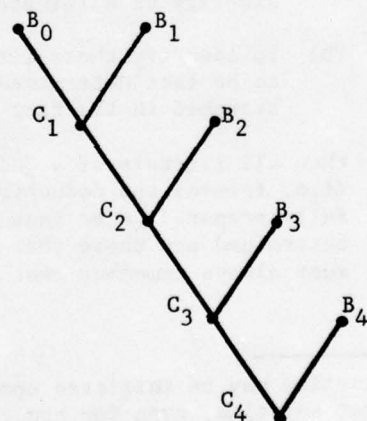
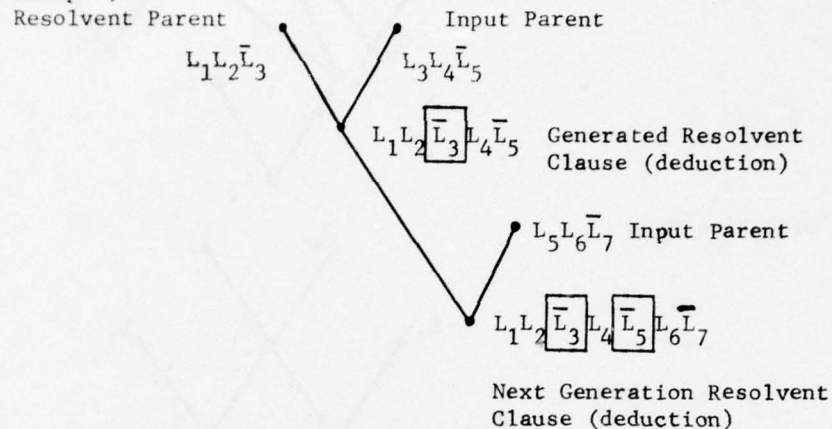


Figure 2-2. Deduction Structures

- (2) Parent clause pairs have their resolution literals positioned at opposite ends of their respective clauses. By convention in OL-deduction, the resolvent parent clause has its resolution literal at the right-most position of the clause.
- (3) The resolution literal is retained in the generated resolvent clause and marked as such by a frame symbol, for post processing, i.e. $\boxed{L(X)}$ means $L(X)$ is a resolution literal.
- (4) The relative positions of the resolution literals with respect to the other literals are preserved in the resolvent clause.

Example,

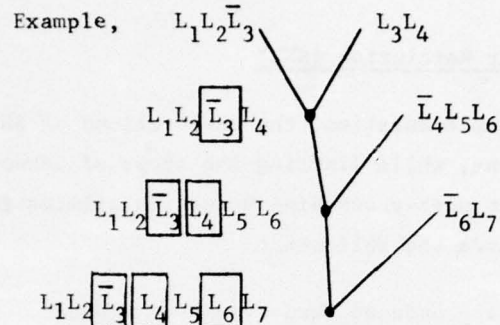


In this example, $\boxed{\bar{L}_3}$ and $\boxed{\bar{L}_5}$ are the resolution literals in the bottom deduction.

- (5) Resolution literals (i.e. framed literals) in resolvent clauses are utilized in two ways.
 - (a) In post processing of a resolvent deduction to simplify or eliminate redundant deductions.
 - (b) To identify those literals in a deduction that are to be fact determined, e.g. for which facts may be searched in the fact base.
- (6) When all literals of a deduction are resolution literals (i.e. framed) the deduction is fully resolved and, therefore, fully prepared to be fact determined.* The facts to be determined are those that refute the framed literals. (We must always remember that we are in a refutation procedure.)

* Fact determination may be initiated upon any or all resolution literals of a deduction at any time, even for non fully resolved deductions. The selection of literals for fact determination is guided by a strategy that minimizes fact retrieval costs and response delay.

The mechanism of the OL-deduction procedure is such that all resolvents reflect their ancestry chain. This is due to the ordering of literals and the retention of resolution literals in the resolvent clauses. As a result, as long as all framed literals are retained and ordering of literals is preserved, one can reconstruct from each resolvent its ancestral deduction chain.



The advantages of the OL-deduction procedure are two-fold.

- (1) It provides a straight forward and orderly procedure for generating deductions, limiting the deduction procedure to (partial) tree rather than a general lattice structure.
- (2) It facilitates certain operations related to minimizing the growth of non-useful deductions - these will be referred to subsequently as deduction simplification and elimination operations.

In limiting itself to the above tree structure, the OL-deduction procedure is incomplete, that is, not all possible deductions can be generated. The completeness of the OL-deduction procedure, however, is assured by a process called "reduction" that is applied to resolvents, see section 2.3.5. "Reduction" in effect permits a resolvent to be created of two parents with common ancestry.

2.3.2 OL-Deduction Compatible with Set of Support Strategy

An added feature of OL-deduction is its relation to the set of support strategy, which tends to avoid development of certain classes of useless deductions.

The set of support strategy states that if within the set of potentially unsatisfiable clauses (facts, rules and negation of query) there exists a subset which is known to be satisfiable (facts and rules), in each resolution select at least one parent clause from outside this satisfiable subset.

In OL-deduction, rule clause parents generally provide a set of support. Clauses taken from the set of negated query clauses are external to the set of support as are unsatisfisfiable resolvent clauses since if at least one parent of a resolvent clause is chosen from outside the satisfiable set, then the resolvent formed is also from outside that set.

2.3.3 Selective Negative Linear Resolution (SNL)

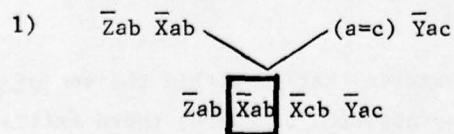
For the first stage of implementation, the restrictions of SNL can be incorporated. These restrictions, while limiting the scope of deduction, also eliminate the need for certain post-processing deduction elimination operations. SNL restrictions include the following:

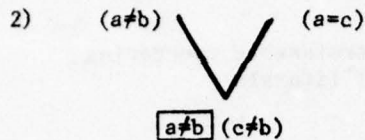
- (1) Input rule clauses are ordered Horn clauses. A Horn clause is a clause containing at most one positive literal. An ordered Horn clause places its positive literal at an end (the left-most) position.
- (2) The top (negated-query) clause of an OL-deduction chain is completely negative, i.e., all literals are in negative form.

As a result of these restrictions, all resolvents in SNL are completely negative. Furthermore, only Horn clauses with a (exactly one) positive literal can enter the derivation since purely negative input clauses cannot resolve with purely negative resolvents.

2.3.4 Paramodulation

Paramodulation is a specialized form of resolution which accepts the equality literal as a special argument. When paramodulation is in effect, there is no need to add explicitly the axioms of transitivity and symmetry of equality. Furthermore, resolution with paramodulation is more efficient than use of explicit axioms. The following examples demonstrate paramodulation.





2.3.5 Deduction Simplification and Elimination: Post Processing Operations

When a resolvent clause has been created, a series of operations are performed upon it to complete resolution, to remove redundancies and to eliminate the clause if it is totally redundant or subsumed by other clauses in the deduction tree. These operations are described in the following:

(1) Factoring

When two unframed unifiable literals of the same sign appear in a deduction clause, it is sufficient to refute the less general of the two, since what refutes the less general literal also refutes the more general one. Dropping the less general literal is called factoring, and is preceded in OL by unification.

Example: $\bar{D}_{3x} \bar{A}_{3x} \bar{B}_{3x} \bar{A}_{3y} \bar{C}_{3y}$ and $X \subset Y$ produces $\bar{D}_{3x} \bar{A}_{3x} \bar{B}_{3x} \bar{C}_{3x}$

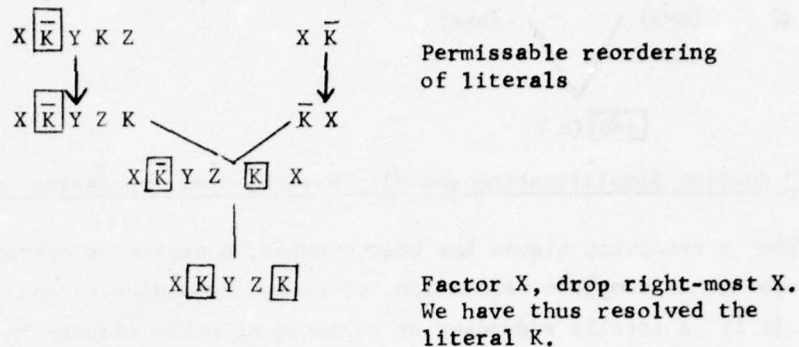
Factoring should always be applied to pairs of literals which are identical. If factoring is forced, however, by fully exploiting all possible unification opportunities, useless deductions tend to be generated.

(2) Reduction without Unification (Not applicable under SNL)

Reduction is a process by which an unframed literal within a deduction is resolved if it is unifiable with a complementary (opposite signed) framed literal to the left of it in the same deduction clause. Reduction without unification is the resolution of a literal which is identical to the framed complementary literal in the clause.

Reduction is in effect resolution of a deduction with another ancestrally-related deduction followed by factoring. For example, consider the clause $X \boxed{K} Y K Z$. Reduction will reduce this to $X \boxed{K} Y Z \boxed{K}$. This can be justified in light of the structure of the given clause which reveals an ancestor clause XK. Reduction in this example can be seen as a resolution of the given clause with the ancestor XK as follows:

Example of Reduction (Without Unification)



(3) Tree Path Subsumption

One clause subsumes another clause if it is more general. If an unframed literal is identical to a framed literal to its left in a deduction clause of a deduction tree, it can be concluded that this deduction is subsumed by a direct ancestor deduction.

Consider the clause $X[\bar{K}]YKZ$. The left side of this clause shows a direct ancestor deduction XK that is more general than the given clause. Since XK subsumes $X[\bar{K}]YKZ$ and is simpler to refute, XK is retained and $X[\bar{K}]YKZ$ is discarded.

(4) Tautology

A tautology is a clause containing two complementary unframed literals (e.g. $P \vee \bar{P}$), is always true and, therefore, cannot be refuted. Further applications of facts and rules can only lead to other true clauses and not to refutation, therefore, tautological deductions are discarded.

(5) Reduction with Unification (Not applicable under SNL)

For each substitution which unifies an unframed literal and a previous complementary framed literal to its left in the same clause, generate a new deduction clause by making the necessary unification substitution and deleting the unframed literal. (The given clause may be retained for other possible resolution operations.)

Example:

$$\bar{A}_{13} \bar{B}_{1x} A_{x3} \text{ produces } \bar{A}_{13} \bar{B}_{11}$$

Another form of reduction which can be performed at this point comes from paramodulation as expressed in the following examples:

- 1) $\boxed{a = b} \overline{Fa}$ produces $\boxed{a = b} \boxed{\overline{Fa}} \overline{Fb}$
- 2) $\boxed{\overline{Fa}} (a = b)$ produces $\boxed{\overline{Fa}} \boxed{a = b} Fb$

Though reduction is defined as a post processing operation, it is, in reality, more of a resolution operation and, therefore, should be executed within the resolution procedure or prior to other post processing operations.

(6) Ordering

Literals to the right of the right-most framed literal are free to be reordered. Reordering may not occur across framed literals without special care.

2.4 FACT DETERMINATION

OL-deduction, as has been stated, is a refutation procedure. A questionable assertion, that is, a query, is presented in negated form to the OL-deduction mechanism. Deductions are then generated and are sought to be refuted. A refutable deduction implies the existence of a collection of rules and facts (possibly a null collection) that refutes the negated query, and consequently, satisfies the assertion of the query. It is the collection of facts that is the answer to the positive query and that is provided by the inference system.

A resolved deduction is a clause consisting of only resolution literals; that is, it is a deduction clause that has been fully resolved.

Each resolution (framed) literal in a resolved deduction clause is a literal about which the information system can supply facts or is an acceptable inferred literal. An inferred literal will often be a simple binary proposition, signifying a yes or no; true or false; 0 or 1 value. In the more general case, where the framed literal is a complex predicate, the literal will tend to be a fact literal for which the information system will supply all facts. For example, given $\boxed{L(X,5)}$ where $L(X,5)$ means "X" is a worker in factory "5", then the information system will search out all workers in factory number 5 and supply these as facts satisfying the literal. We call this procedure "fact determination". When fact determination is applied to all of the fact literals of a completely resolved deduction, we then determine a largest subset of these facts that can satisfy the deduction as an entirety.

The size of this subset of facts is dependent on the way in which the resolution literals, and their terms, of the deduction clause are interrelated.

Example: $\boxed{\bar{L}_1(X,5)}$ $\boxed{\bar{L}_2(X)}$ are two interdependent literals in a deduction with the meanings

$L_1(X,5)$, X are men working in factory 5

$L_2(X, < 45)$, X are men under 45 years of age

The set of facts refuting the deduction is the logical intersection of the sets of facts refuting the two framed literals independently.

2.4.1 Summary of Sources of Facts for Deduction Evaluation

The sources of facts for deduction evaluation are as follows:

- (1) Fact Base of the information system. This is the primary source of fact data. The fact base consists of data bases and relational data sets.
- (2) Fact Generator Algorithms. Algorithms that generate facts range from simple to complex. Simple algorithms are executed by simple computer programs, some of which may be under direct control of the inference system. Such algorithms may perform simple range filtering, or selection tests for some specific literals. (Fact Generator Algorithms are an open area for further investigation and are only briefly touched upon in this report.)
- (3) Users. Users via dialogue may contribute facts to facilitate the inference processing.
- (4) Facts generated by deduction. These are inferred facts, not verifiable (satisfiable) by the information system.

2.4.2 Deduction and Verification: A Two Stage Procedure

The deduction verification procedure may be viewed as consisting of two parts, a deduction development stage called Stage 1 and a deduction verification or fact search stage called Stage 2. This is depicted in Figure 2-3.

Stage 1 generates deductions by resolving parent deductions, queries, and rules from the rule base. Stage 2 acts to acquire facts that will satisfy the resolved literals of a deduction.

The first stage preserves the interdependence of literals within deductions. The second stage takes into account this interdependence by a procedure of fact "intersection". For example, in the clause $\bar{A}x\bar{B}x\bar{C}xy\bar{D}yz$, all of the literals are inter-related by their terms. If this deduction is in a form for fact base instantiation, then it proceeds to Stage II where fact instances (as many as are required) are sought that refute it.

The intersection operation could be a lengthy, costly procedure, since it involves information retrieval from the fact base and set intersection operations. To minimize the work involved in the intersection process and to reduce the processing time, optimal intersection strategies may be employed. We illustrate these two stages with the following example.

Example:

- (1) State 1 passes to Stage 2 the inter-related literals of a deduction $\bar{A}x\bar{B}x\bar{C}xy\bar{D}yz$, for fact retrieval and intersection.
- (2) Stage 2 requests and receives the following facts in data files $\{Ax\}$, $\{Bx\}$, $\{Cxy\}$ and $\{Dyz\}$
- (3) Stage 2 computes the intersection domains x , y and z by the following (or some other intersection) procedure.

$$\begin{array}{l} \{X_i\} \leftarrow \{A_x\} \wedge \{B_x\} \wedge \{C_x\} \dots \text{intersection over subscript } x \text{ only} \\ \{Y_{ij}\} \leftarrow \{C_{x_i}y\} \wedge \{D_{yz}\} \dots \text{intersection over subscript } y \text{ only} \\ \{Z_{ij}\} \leftarrow \{D_{y_{ij}}z\} \end{array}$$

- (4) Stage 2 then sends the domain values $\{X_i\}$, $\{Y_{ij}\}$, and $\{Z_{ij}\}$ to Stage 1.

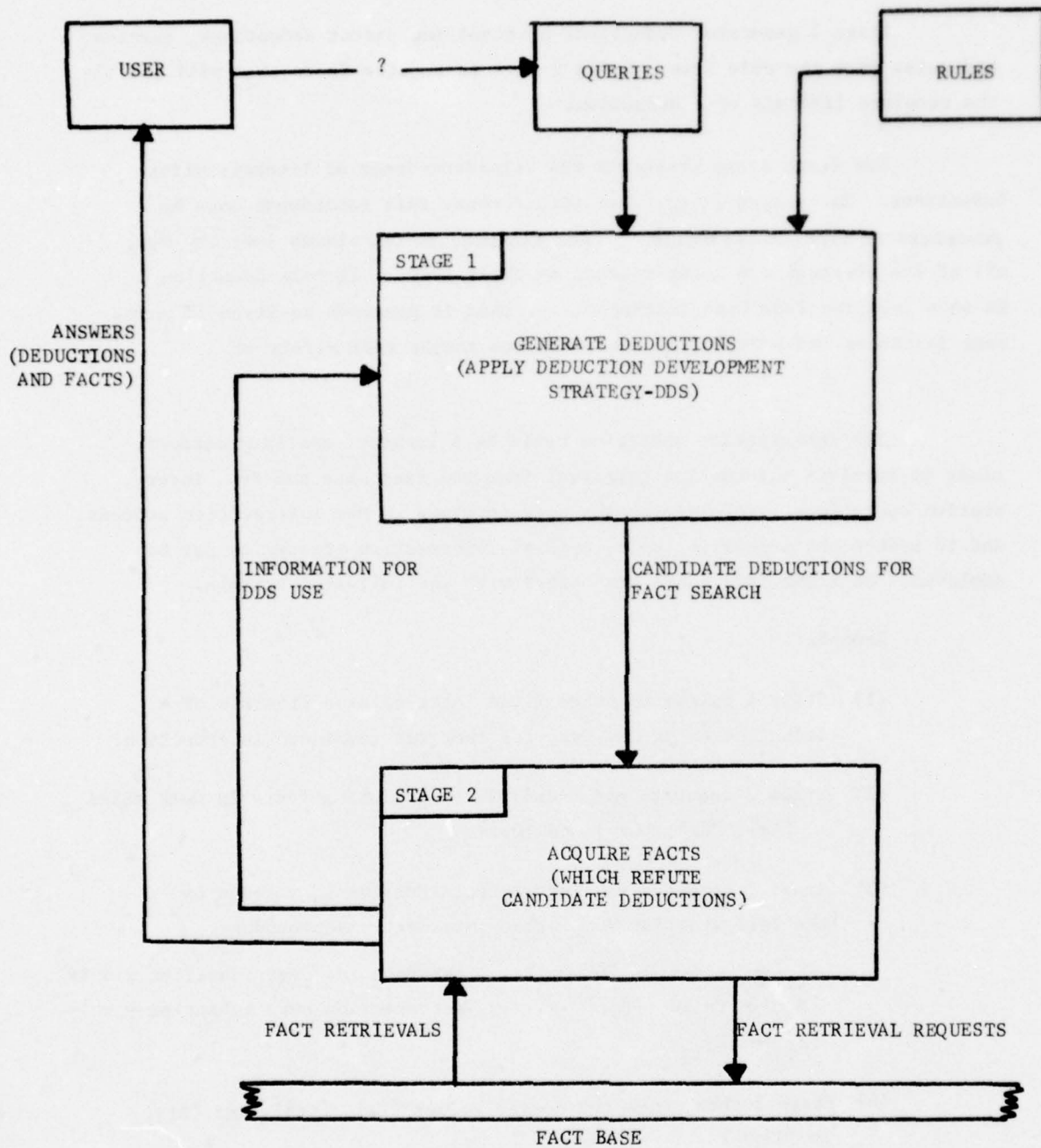


Figure 2-3. Deduction and Verification Stage 1 - State 2 Procedure

2.4.3 Effect of Actual vs. Statistical Fact Data on Deduction Development

In the Stage 1 - Stage 2 operation, Stage 2 is able to withhold all fact information from Stage 1 until the final returns for related literals arrive from the fact base. Stage 2 can then perform its intersection data operations. Delays may arise, out of Stage 2's control due to fact retrieval delays. As a result, Stage 1, which operates very rapidly, may have generated a number of deductions beyond what was needed, since, had Stage 1 received fact data earlier, it might have been able to conclude its operations earlier with possibly a smaller deduction tree. Also, Stage 1 does not require actual facts, but can proceed simply with information about the existence of facts. This information may precede actual arrival of facts from the information system, or may be available to the inference system as locally maintained data, estimated on the basis of past system performance. Thus, Stage 1 could proceed with little or no actual facts from Stage 2 for most of its tree development. This, incidentally, also means that Stage 1, prior to receiving actual fact data, generates the same deduction tree regardless of changes in the fact base, so long as the information about the facts (e.g., domains and performance statistics) have not been (substantially) changed by an update. When Stage 1 receives current information about facts from Stage 2, it will develop its deduction tree using historical (performance statistics) information. The extent that Stage 2 may delay, for the advantage of efficient fact (batch) retrieval and intersections, must be traded against the sluggishness such delay causes in Stage 1 performance.

SECTION III. DEDUCTION DEVELOPMENT STRATEGY

An objective of the Inference System is to minimize the likelihood of generating a large number of deductions in the search for a query refutation. To meet this objective, the following operations are applied:

- (1) Context Set Formation
- (2) Deduction Selection/Ordering Strategy
- (3) Deduction Tree Pruning

3.1 CONTEXT SETS

A Context Set in an inference system is a consistent set of related axioms (rules and facts) and queries. The axioms of an inference system form one or more independent Base Context Sets. When a query clause is added to the axioms of an inference system, a new context is formed consisting of the query clause and all of the base context sets to which the query clause is related. Clauses are related when they have common literals.

Example:

(1) Base Context Sets $\{BCS\}$ of the Inference System:

$$BCS_I \begin{cases} Axy \bar{B}xz \ Cz \\ Bxz \ \bar{C}z \\ Cz \ \bar{F}z \\ Fz \end{cases} \quad BCS_{II} \begin{cases} Gxy \ \bar{H}xy \\ Hxy \ \bar{I}xy \\ Ixy \end{cases} \quad BCS_{III} \begin{cases} Kxy \ \bar{K}xz \ \bar{K}zy \\ Kxy \ \bar{L}xy \\ Lxy \end{cases}$$

(2) Query: $\bar{A}xy \ \bar{B}xz \ \bar{G}xy$

(3) New Context Set: Query + $BCS_I + BCS_{II}$

When a query is to be refuted, it is not necessary to consider the unrelated base context sets of the inference system. Our motivation in the examination of context sets is to determine how to minimize the size of context sets (they can always be enlarged) in the expectation that the smaller context sets provide a better (quicker, cheaper) solution to an inference problem.

3.1.1 Context Set Formation

A context set is a set of axioms, query clauses and an associated set of literals. The set of literals includes all literals in the axioms and query clauses of the context set. Two context sets are independent if they do not have literals in common. We define "base context sets" as context sets containing axioms only, and apply the broader expression "context set" to include query clauses. A clause is understood to be a disjunction of one or more literals.

We define a Context Set formally by:

$$\text{Context Set} = \{L\} \cup \{A\} \text{ where for each } L_i \in L; \text{ and, } A_j \text{ and } A_k \in A, j \neq k; \\ [L_i \in A_j \Rightarrow \bar{L}_i \in A_k] \text{ holds.}$$

where L refers to Literals and A refers to axioms and query clauses.

In the inference system, we start with a set of axioms and generate a set of independent base context sets. When a query clause is entered into the inference system, an independent context set is formed by combining those basic (independent) context sets that share literals with the query. However,

since the deduction mechanism operates on one query literal at a time (see OL-deduction procedure), the deduction procedure will actually alternate from one base context set to another base context set according to the context membership of the query literal currently being resolved. The literals of a query clause can, of course, be arranged in groups according to their base context set membership to minimize context hopping.

3.1.2 Context Set Hierarchy

A context set may be further subdivided into a collection of weakly related context sets, in such a way as to form a hierarchy of related context sets. The highest rated context set includes the query(s) to be refuted; the lower rated context sets of the hierarchy contain axioms only.

One way of creating a context hierarchy involves ascertaining and assigning weight values to the literals with respect to any particular query. The weight values are best determined and provided by the inquirer who may have external knowledge as to the relative importance of literals toward resolving his query. A secondary source of weights may be supplied by the inference system, computed on the basis of operational statistics. Determination of weights for context formation is discussed in paragraph 3.1.5; weighted context sets and context bridges are discussed in paragraph 3.1.3.

3.1.3 Context Set Weights and Bridges

When weights are assigned to the literals of a context set, they may be used against a set of threshold values to subdivide the context set into a hierarchy of two or more levels. The resulting context sets in the hierarchy are linked by axioms, which we shall call axiom bridges, which contain literals in more than one of the hierarchy's context sets. The following example illustrates the formation of a hierarchy of context sets.

Example: Given a Context Set with the following literals, weights and axioms.

Axioms	R1 = $\bar{A} \bar{B} \bar{C} D$	Literals	A	Weights	High
	R2 = $B C \bar{D} F$		B		High
	R3 = $C \bar{E} F$		C		High
	R4 = $D \bar{G}$		D		High
	R5 = $E \bar{F} \bar{A}$		E		Medium
	R6 = $\bar{F} A \bar{B}$		F		High
	R7 = $G \bar{C} \bar{F}$		G		Low

Using the three weight thresholds - High, Medium and Low - we generate the three context set divisions:

Context Sets (Divisions)	Literals	Axioms	Weight
I	A,B,C,D,F	R1, R2, R6	High
II	E	R3, R5	Medium
III	G	R7, R4	Low

In this example, the axioms of context sets II and III include literals in context set I. In general, each context set in the hierarchy bridges upwards, that is, the set of axioms of any level (but the highest) will contain one or more literals of one or more context sets above it. A good context set hierarchy construction would display "narrow-weak" bridges ("narrow" suggests few upward linking literals in the rules; "weak" suggests that link literals have low weights).

Note also that each literal is included in its axiom set or higher level axiom set at least once in both its complemented and uncomplemented form.

3.1.4 Application of Context Sets

When literal weights are available, context sets can be narrowed by subdividing them into a hierarchy of context set levels. The highest level context set is applied first, and, when its possibilities have been exploited,

the next level context set is applied and so on. This progression expands the context set from its top level to successive inclusions of its lower levels. The expectation is that most applications will be satisfied within the higher levels of the hierarchy.

3.1.5 Deriving Context Weights for Literals

The best way of acquiring literal context weights is by having these supplied by the inquirer, since he may have considerable knowledge concerning the context of his query that is not contained in the inference system. At the other end, the system can maintain statistical data for computing probability weights. This data can be pre-computed periodically, based upon actual performance results and upon simulation runs designed to derive these weights.

A simple and straightforward method of deriving literal weights for a context set hierarchy formation is to tabulate the number of uses of each literal of the context set over a period of time. A literal "use" can be defined as an instance when the literal was involved in a success path in a deduction tree. An example is presented in Table 3-1.

3.2 DEDUCTION ORDERING AND SELECTION FOR DEDUCTION DEVELOPMENT

In this section we discuss the criteria used for deduction development. We begin with the ordering of literals within a clause, since, in OL-deduction the order of literals in a clause determines the order of resolution of that clause with other clauses to form deductions. We then discuss the order of selecting deduction clauses for deduction development.

3.2.1 Literal Ordering

In deduction refutation, all of the literals of a deduction clause are to be refuted if the refutation procedure is to terminate. The order in which these literals are to be refuted is not logically constrained. If the deduction is not refutable, it is best to learn this as soon as possible, suggesting, therefore, as a first criterion, that the literals be processed in the order of their least likelihood of refutation.

TABLE 3-1

EXAMPLE OF LITERAL WEIGHTS DERIVATION

<u>Context Set</u> <u>Literals</u>	<u>Number of</u> <u>Successes</u>	<u>Thresholds</u> <u>low < 10 < medium < 50 < high</u>
L1	1	Low
L2	50	Medium
L3	100	High
L4	2	Low
L5	30	Medium
L6	20	Medium
L7	100	High
L8	50	Medium
L9	79	High
L10	5	Low

Context Hierarchy

<u>Level</u>	<u>Literals</u>	<u>Weight</u>
1	L3, L7, L9	High
2	L2, L5, L6, L8	Medium
3	L1, L4, L10	Low

A second criterion for literal ordering within a deduction for deduction refutation, is to attempt the easiest to refute literals first. This criterion, following the first criterion, gives the best (least cost) expectation of encountering a non-refutable literal.

A third criterion is to process literals of deduction in an order that minimizes operations and computational burdens. As will be seen, this criterion is particularly applicable to interdependent literals.

3.2.1.1 Least Likelihood of Refutation Criterion for Literal Ordering in a Deduction

The first criterion for ordering literals in a deduction selected for refutation, is to attempt the least likely to refute literals first.

The only prior knowledge leading to a recognition of the refutability of literals in a goal is probabilistic. Thus, probability values can be assigned to each literal in the inference system that indicates the likelihood of its refutability. The probability values could be user supplied or could be computed on the basis of statistical performance data involving the literal, over a period of time. The performance statistics collected could include:

- (A) Number of occurrences (uses) of the literal in deductions,
- (B) Number of failures to resolve the literal in deduction development,
- (C) Number of failures to satisfy the literal with facts,
- (C1) Deduction clause characteristics: Number of literals; Number of terms (max, avg),
- (C2) Term characteristics: Constants; variable simple, complex.

A probability formula indicating the likelihood of refutability of a literal may be defined by:

$$P_L = \frac{A_L - B_L - C_L(C_1, C_2)}{A_L} = \text{Probability of Literal}$$

The function $C_L(C_1, C_2) \leq C_L$, depends on the values of C_1 and C_2 .

The statistical data collection is a background activity and, of course, need only be updated periodically.

The probability formula could be made more relevant by improving statistics collected to show relationships among literals, for example, replacing (A) and (B) by:

(A') Number of co-occurrences of L_a and L_b ($b \neq a$) in a deduction when L_a and L_b are, in fact, related by their terms, e.g., $L_{xy} C_y$.

(B') Number of failures to refute L_a under condition (A') above.

This would take into account, to a greater extent, the inter-dependency among literals in a deduction.

The computed probabilities would also have computed confidence levels that would, in effect, define the precision of the probability estimates. The probability values would then be utilized as the first criterion for ordering the literals of a deduction. The effectiveness of this criterion depends upon the quantity and quality of the statistical data and the storage cost for maintaining it. (The off-line processing cost should be a secondary consideration.)

3.2.1.2 Easiest to Refute Criterion for Literal Ordering in a Deduction

The easiest to refute criterion for ordering the literals in a deduction for refutation focuses on the work required to refute a literal. This criterion is greatly clarified by the separation of the deduction refutation procedure into two stages, described in 2.4.2. The first stage deals substantially with deduction tree development and the second stage deals substantially with fact determination.

3.2.1.2.1 Stage 1, Deduction Development: Literal Ordering Criteria

(1) Fact Literals - First Criterion

Literals in the deduction which are only resolvable by the fact base are given the highest priority since there is nothing that has to be done with them in Stage 1. Such literals are passed on to Stage 2.

(2) Deduction Development Effort - Second Criterion

Each literal in the system can have determined for it a merit value representing the effort statistically associated with its refutation. High merit literals are processed before literals of lower merit. A literal's merit (effort) value can be determined as a function of the average deduction subtree size statistically generated under the literal (as the root of the subtree), over a period of time. The statistics and merit values can be computed as a background process.

(3) Independent Literals - Third Criterion

Here we may look ahead to Stage 2, and organize the literals according to interdependent sets. The set of literals with the least interdependence may be ordered into first positions on the presumption that where there is the least interdependence we will have the least work in Stage 2 for instantiation.

(4) Projected Fact Search - Fourth Criterion

An estimate can be made of the fact search effort associated with a literal, based upon fact base statistics for the literal. This requires maintaining statistics in the fact base on literals $\{L(x,y,z,...)\}$ such as:

- (1) Maximum number of entries in the domain $L(x,y,z,...)$
- (2) Average number of entries in the lesser domains

$L(-,y,z,...)$
 $L(x,-,z,...)$
 $L(x,y,-,...)$

The literals with the smallest range, presumed easiest to process, are placed first in the remaining ordering of literals in a deduction.

(5) Interdependent Literals - Fifth Criterion

Literals with the least dependence in an interdependent group are placed first. For example, $Axy Bx3$ rather than $Bx3 Axy$, where the right-most is the first position. Also, $Xxy3 Bxy Cz$ rather than $Cz Bxy Axy3$ or $Bxy Xz Axyz$.

3.2.1.2.2 Stage 2, Fact Determination: Ordering for Fact Retrieval and Intersection

Stage 2 ordering applies to the "fact determination process" of deduction refutation. Here we are concerned with determining the domain of

the literals available in the fact base, and their intersection for inter-related literals of a deduction.

A special strategy and a general approach to determining the best order for fact retrieval is shown in Figure 3-2. The special strategy chooses the literal, of a clause of interdependent literals, with the least cost to fact retrieve. This might well be the literal in the clause with the least fact range. When this literal is fact retrieved, its domain will be determined. In the example of Figure 3-2, Axy is the first, least cost estimated literal chosen; and the domain terms x, y are determined. This information enables us to recompute more finely the costs for fact retrievals of the remaining literals. A least cost literal is now chosen from each of the remaining interdependent sets of literals, for fact retrieval. In this example, the literals are Bxz and Cy, which are independent of each other. The process is repeated until all of the literals of the original clause have been fact retrieved. In this strategy, an intersection of facts is automatically performed, while attempting to minimize the quantity and cost of fact retrieval. This strategy, while appealing, is not optimal. An optimal procedure involves full development of a decision tree, such as the and-tree shown in Figure 3-2, and a cost estimate of each of the root to link tree paths. The least cost path is the best. As described thus far, this approach provides a retrieval ordering plan that, of course, could turn out to be suboptimal since it depends upon estimates of performance and retrieval costs. It could be improved by recomputing costs over the remaining subdecision tree after each fact retrieval action; or, more generally, by restructuring, after each fact retrieval action, a full decision tree for the remaining literals including using updated retrieval costs.

The general approach would be costly to compute for very large clauses of interdependent literals, but not very costly for small clauses (say less than 6 to 8 literals).

When Stage 2 receives requests from Stage 1 for literal fact retrievals, it may receive these literals one or more at a time. Stage 2 will determine the way in which these retrieval requests will be passed on to the fact base for the actual retrieval. Stage 2 may issue an order immediately or wait a fraction of a second or more, in which time Stage 2 may receive

Example: Retrieve and Intersect Facts to Satisfy Axy Bxz Cy Dz

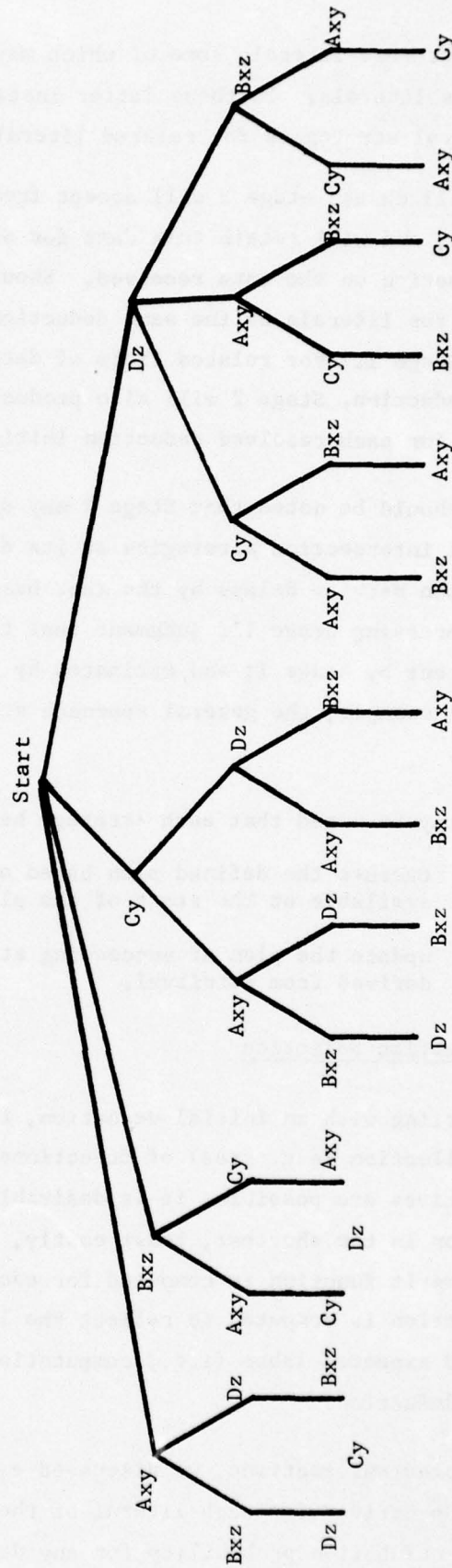
I. Special Strategy

Cost
C(Axy)
C(Bxz), C(Cy)
C(Dz)

- (1) Choose Min (Axy, Bxz, Cy, Dz) = Axy
- (2) Choose Min (Bxz, Dz) Min (Cy) = Bxz, Cy
- (3) Choose Min (Dz) = Dz

Cost 1 = Cost (Axy, Bxz, Cy, Dz)

II. General Approach - Try all Combinations choose min path cost



3-11

Compute Cost of Paths and Choose Smallest Cost Path

Figure 3-2. Methods of Fact Retrieval Ordering and Intersection

from Stage 1 yet more literals some of which may come from the same deductions as the previous literals. In these latter instances, Stage 2 can perform optimal retrieval strategies for related literals of the same deduction.

In all cases, Stage 2 will accept from the fact base lists of fact data retrieved, and will retain this data for Stage 1, but will report to Stage 1 summary information on the data received. Should Stage 2 receive two or more lists of data for literals of the same deduction, this will also be noted and passed on to Stage 1. For related lists of data, pertaining to related literals of the same deduction, Stage 2 will also produce an intersection list which it will maintain for each resolved deduction initiated by Stage 1.

It should be noted that Stage 2 may elect to execute any one of the fact retrieval intersection strategies at its disposal. The decision factor will depend upon service delays by the fact base, Stage 1 urgency indicators, indicators expressing Stage 1's judgment that the order of retrieval should be the order sent by Stage 1; and, estimates by Stage 2 of the cost of each strategy - for example, the general approach strategy might be excluded for large clauses.

It may be noted that each strategy has two modes of execution,

- (1) execute the defined plan based only upon statistical data available at the start of the plan's execution,
- (2) update the plan at succeeding steps based on current data derived from retrieval.

3.2.2 Deduction Selection

Starting with an initial deduction, the deduction refutation process produces a collection (e.g. tree) of deductions that are candidates for refutation. Since alternatives are possible, it is desirable to select deductions that lead to a refutation in the shortest, least costly, way. To guide the best selection, a merit function is computed for each deduction on the tree. The merit function is computed to reflect the likelihood of a successful refutation and expected labor (i.e., computational cost) associated with refuting the deduction.

In previous sections, we discussed a probability of refutation, which was to be derived for each literal of the inference system. This data could yield a refutation probability for any deduction, i.e., the deduction

refutation probability equals the product of the refutation probabilities of its literals. The most likely to succeed deduction (highest refutation probability), would be considered the most preferred deduction to develop.

Following this first criterion, deductions would be chosen on the criterion of the least effort to process. Also, in a previous section, we identified a means for estimating deduction development costs based on tree development and projected fact search costs associated with literals. We could use these costs to determine the costs of processing a deduction, i.e., expected cost of a deduction is the sum of the expected costs of processing its constituent literals.

NOTE: A budget is established for the number of deductions to be processed at each step of tree development. The budget is based upon consideration of computer storage and processing availability at the time of deduction selection. If n deductions are budgeted, then the first n highest merit deductions are selected from the tree.

3.2.3 Rule Selection for Deduction Development

When a clause is selected for a further deductive development, it is capable of spawning a number of deductions. The number of possible deductions that can be spawned depends upon the number of different rules and facts in its context set, against which the deductions can be resolved. In addition, several constraints may be applied including:

- (1) budget for the deductions,
- (2) a limit on the number and constraints on the selection of deduction literals to be resolved at each development attempt.

To determine which rules are to be resolved against, under a limited budget, two considerations apply:

- (1) selection of the deduction literal to be resolved,
- (2) selection of the rules for resolving, where more than one rule is applicable.

The first consideration will have already been taken care of in a preceeding operation where deduction literal ordering is established. For the second consideration, the applicable rules will have merit values associated with them, and the first n highest merit valued rules are applied. Rule merit values could be computed in the same manner and on the same basis as described above for deduction merit functions except that the weight of the resolving literal need not be included. The merit value of the newly generated deduction is recomputed as the composite merit values of its parents minus the merit value of the resolved literal. To give the most accurate value, the rule merit is computed after the operation of unification.

A final deduction merit value is determined after all operations upon a deduction have been completed, since some of these operations drop literals in the deduction (e.g., factoring) or otherwise modify them. Literal ordering within a deduction is the final step in deduction processing. (In the OL refutation procedure, only the augmenting literals need be ordered since older literals in the deduction once ordered retain their order.)

A refinement of the above method of rule selection is to develop merit values for a rule as a whole, with each merit value of the rule being associated with a literal of the rule that might be used as a resolver. The merit values of the rule could be defined for each of its literals used as resolvers, as the ratio of the number of instances the rule was in a success path divided by the number of times the rule was used, over a period of time.

3.3 DEDUCTION PRUNING

Pruning refers to deleting non-useful deductions on a deduction graph. It is applied whenever a non-useful deduction is derived. Pruning is applied when merit values are ascertained to be below certain established thresholds. It is also applied in deduction post-processing operations involving subsumption determination, tautology, and where certain special deduction graph relations arise.

SECTION IV. PROGRAM DESIGN

4.1 PROGRAM MODULES AND INFORMATION FLOW

Figure 4-1 presents the major flow of information among the program modules of the inference system. Data sets and communications are presented in the figure with the program modules. The principle data sets are the Deduction Tree(s), Rules & Literals File, Fact Information File, Answers File and Performance Statistical File. Communications data include query communication, fact retrieval orders, deduction development orders and tree pruning orders.

Figure 4-2 presents flow chart conventions used to describe program modules. Figure 4-3 to Figure 4-7 highlight the functions of the individual program modules.

4.1.1 Query Module

The Query Module, Figure 4-3, accepts a query input, translates it into a refutation form, initiates a query job request to the Operating Systems Module, and then sets up (initializes) a deduction tree with the negated query clause as top node in the tree. The Query Module also accepts

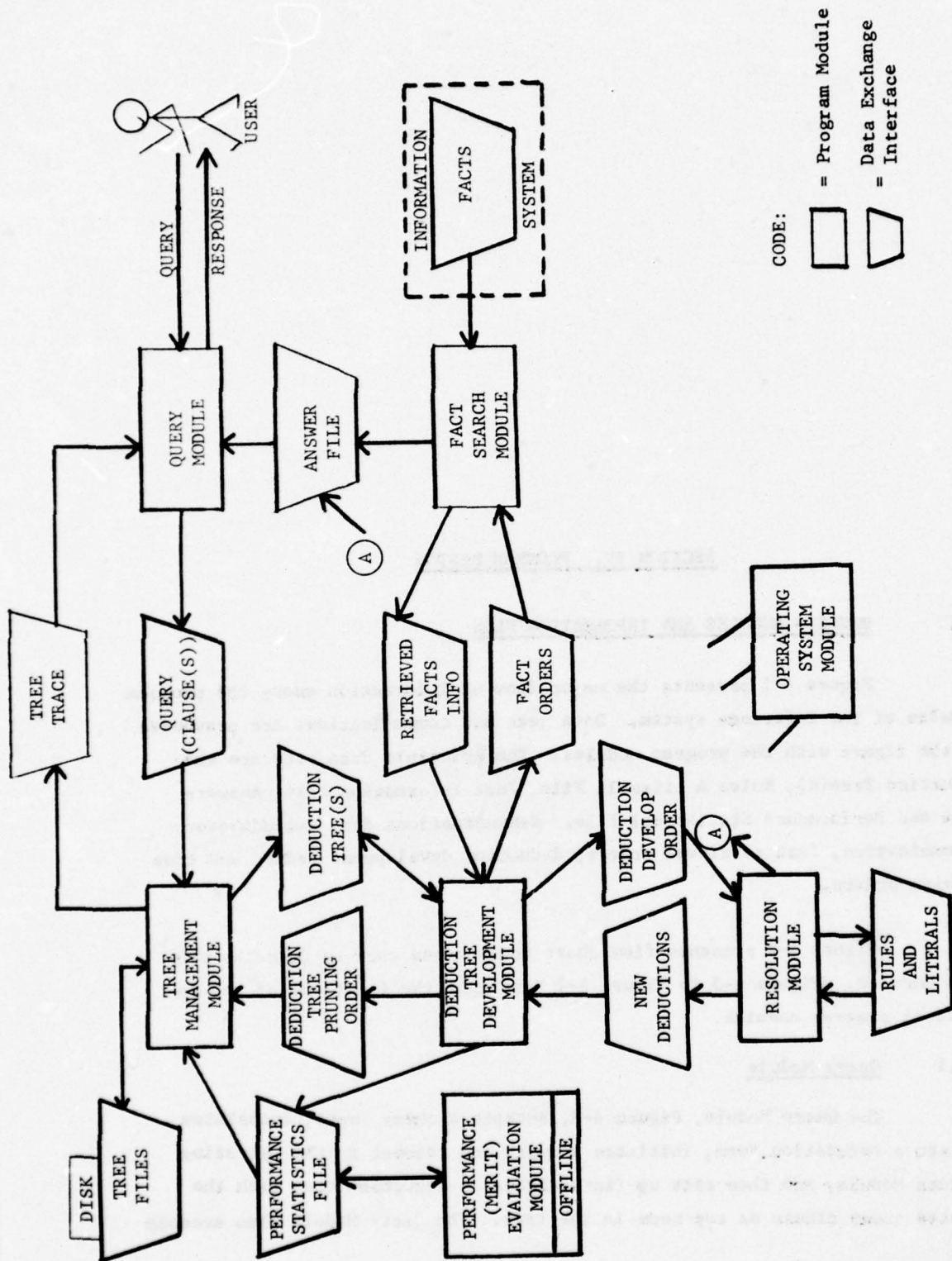


Figure 4-1 Program Architecture
(Modules & Data Flow)

Flow Chart Conventions

These flow charts show process transactions between (or among) data sets (files, tables, messages). Graphic elements of the flow charts are:

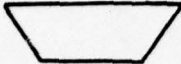

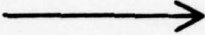
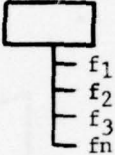
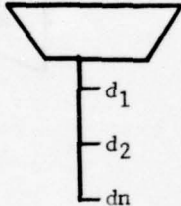
- (1)  trapezoid specified a data set
- (2)  rectangle. Specifies a data transformation, or process.
- (3)  directed flow line. Specifies data flow.
- (4)  same as (2) above, except that functions {fi} of the process are listed outside of the rectangle
- (5)  same as above, except that data item {di} of the data set are listed outside the trapezoid

Figure 4-2 Flow Chart Conventions for Program Modules

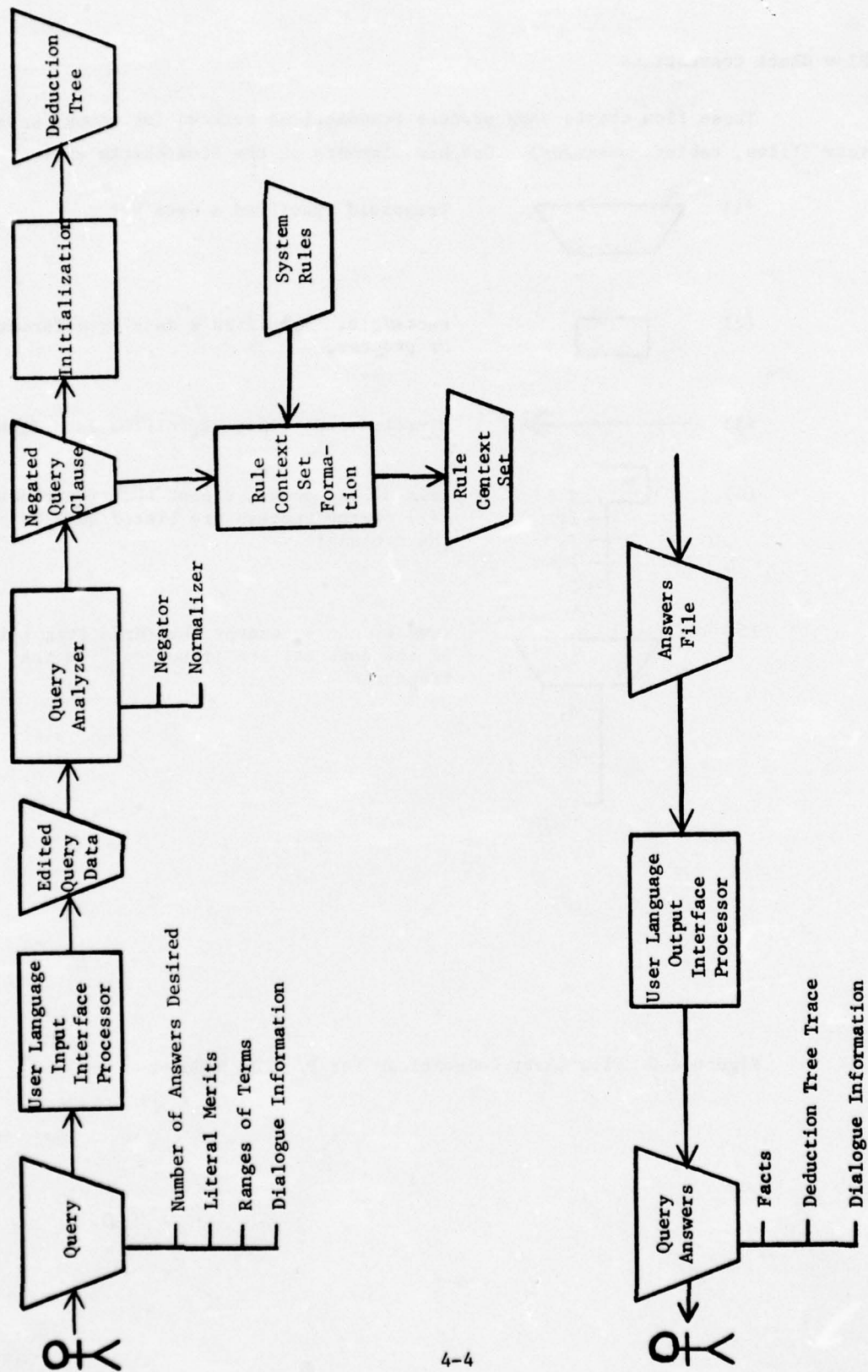


Figure 4-3 Query Module

user data with user recommendations on the context set and on the range value and merits of literals in the query and in the context set. This information is used to determine the query context set and to (re)compute literal merit values. The Query module also interfaces with the user and the Answers file to supply results and aid in user dialogues.

4.1.2 Deduction Tree Management Module (DTMM)

The Deduction Tree Management Module, Figure 4-4, maintains all deduction trees and performs a number of tree processing operations. These include tree pruning operations, such as deletion of inactive, low merit, and subsumed deductions, tracing a tree from specified tree leaves back to the origin query (for report to user to display the deduction sequence), collecting tree statistics (e.g., size, cost of deduction), and other operations for the computer management of the trees.

4.1.3 Deduction Development Module (DDM)

The Deduction Development Module, Figure 4-5, determines the development of deductions. A Deduction Merit Ranking Table in Figure 4-5 contains for each deduction tree, a list of all currently generated deductions of the tree, in order of their highest to lowest merit. The DDM collects queries at initialization and newly generated (resolved) deductions from the Deduction Resolution Module, computes their merits and sorts them into the ranking table. Upon execution, by the scheduler of the Operating System Module, the DDM selects the top m entries of the ranking table as candidates for spawning new deductions (" m " is a quantity to be specified at initialization of the system). These are formed into a work order and sent to the Deduction Resolution Module for further deduction development. Supporting these operations are various other system tables containing fact and literal information.

When a deduction can no longer spawn new deductions, it is deleted from the ranking table. Also, when new information is supplied to the DDM that affects the merit of deductions in the ranking table (e.g., current fact range information), these are recomputed and resorted into the ranking table.

When the ranking table gets too large, if it is not extended on disk storage (by the DTMM), the lowest merit entries are deleted. When deleted

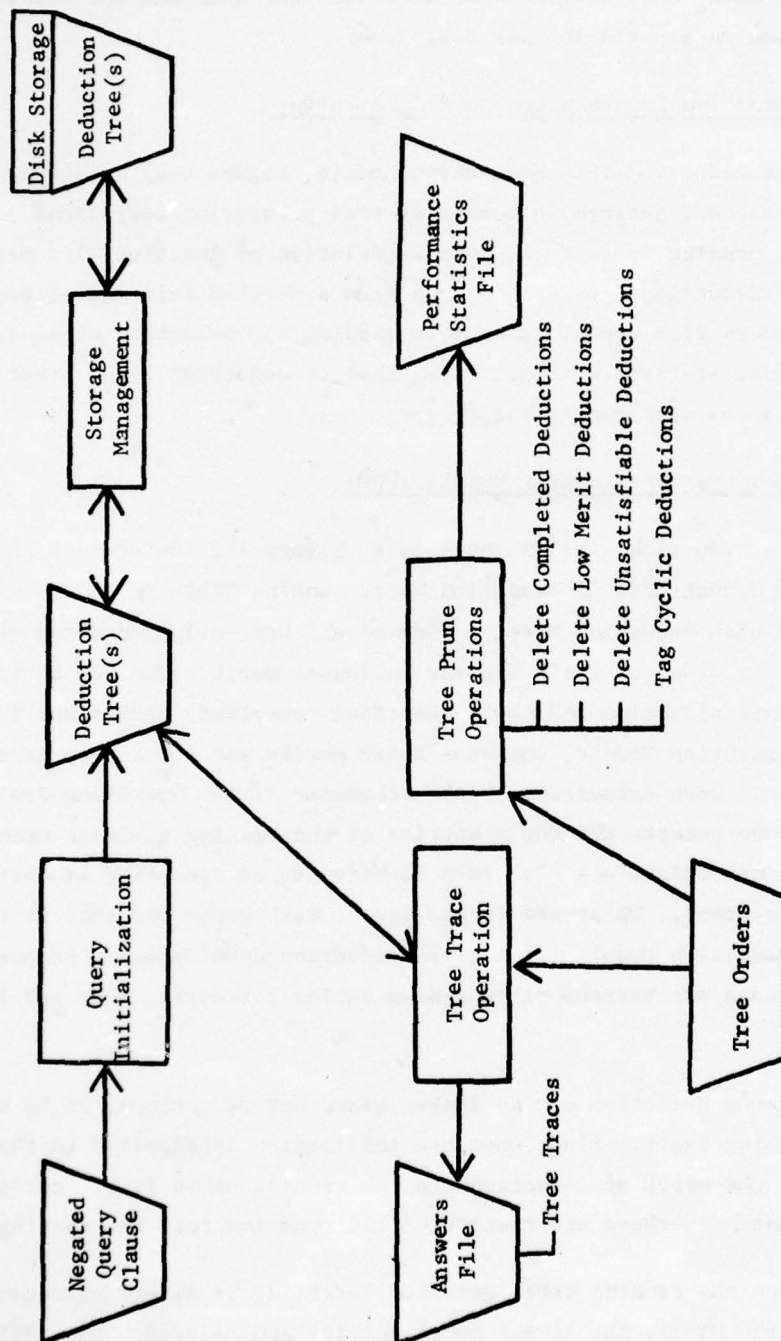


Figure 4-4 Deduction Tree Management Module

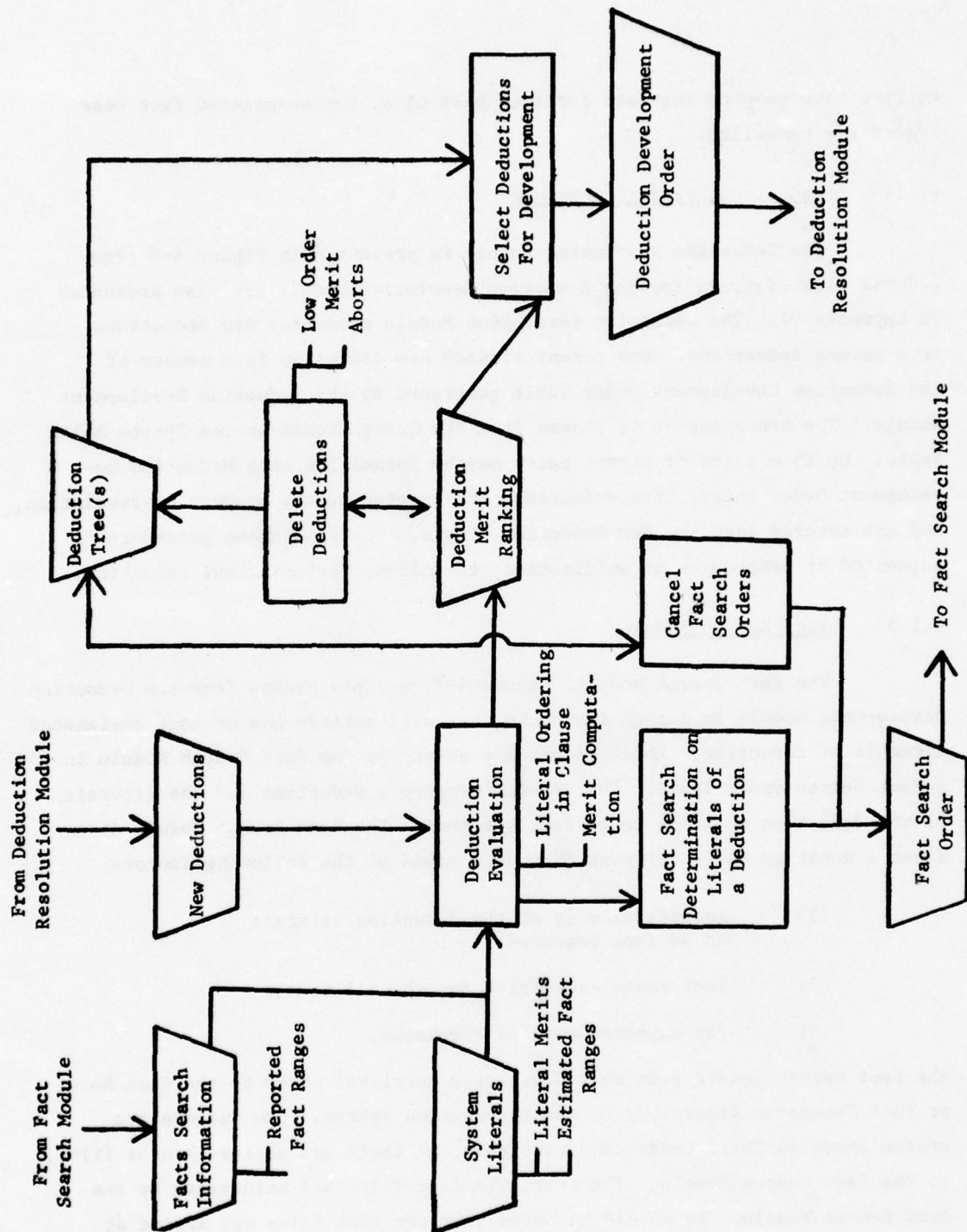


Figure 4-5 Deduction Development Module

entries have pending requests for fact base data, the associated fact base orders are cancelled.

4.1.4 Deduction Resolution Module

The Deduction Resolution Module is presented in Figure 4-6 (Procedural flow diagrams for the Deduction Resolution Module are also presented in Appendix D). The Deduction Resolution Module generates new deductions from parent deductions. One parent of each new deduction is a member of the Deduction Development Order Table generated by the Deduction Development Module. The other parent is chosen from the Query Clause or the System Rules Table. Up to m pairs of parent pairs may be formed for each Deduction Development Order entry. New deductions are created by the process of resolution, and are entered into the New Deductions Table. The resolution procedure is augmented by procedures of unification, tautology, factoring and reduction.

4.1.5 Fact Search Module

The Fact Search Module, Figure 4-7, accepts orders from the Deduction Development Module to search for facts that will satisfy one or more designated literals of deduction. These orders are passed to the Fact Search Module in a Fact Search Order Table. The orders identify a deduction and the literals of the deduction that are to be fact searched. The Fact Search Module determines a strategy for retrieving fact data based on the following factors:

- (1) interdependency of the deduction literals to be fact searched
- (2) fact range statistics for these literals
- (3) the expected cost of retrieval.

The Fact Search Module sets up and passes a retrieval order to the Fact Base or Fact Generator Algorithms of the information system. The information system executes these orders and retrieves the facts and passes them as files, to the Fact Search Module. The retrieved fact files are maintained by the Fact Search Module. It should be noted that the fact files may arrive at uncertain times. Dependent fact files related to interdependent literals of a deduction are then intersected to form fact sub-files satisfying these

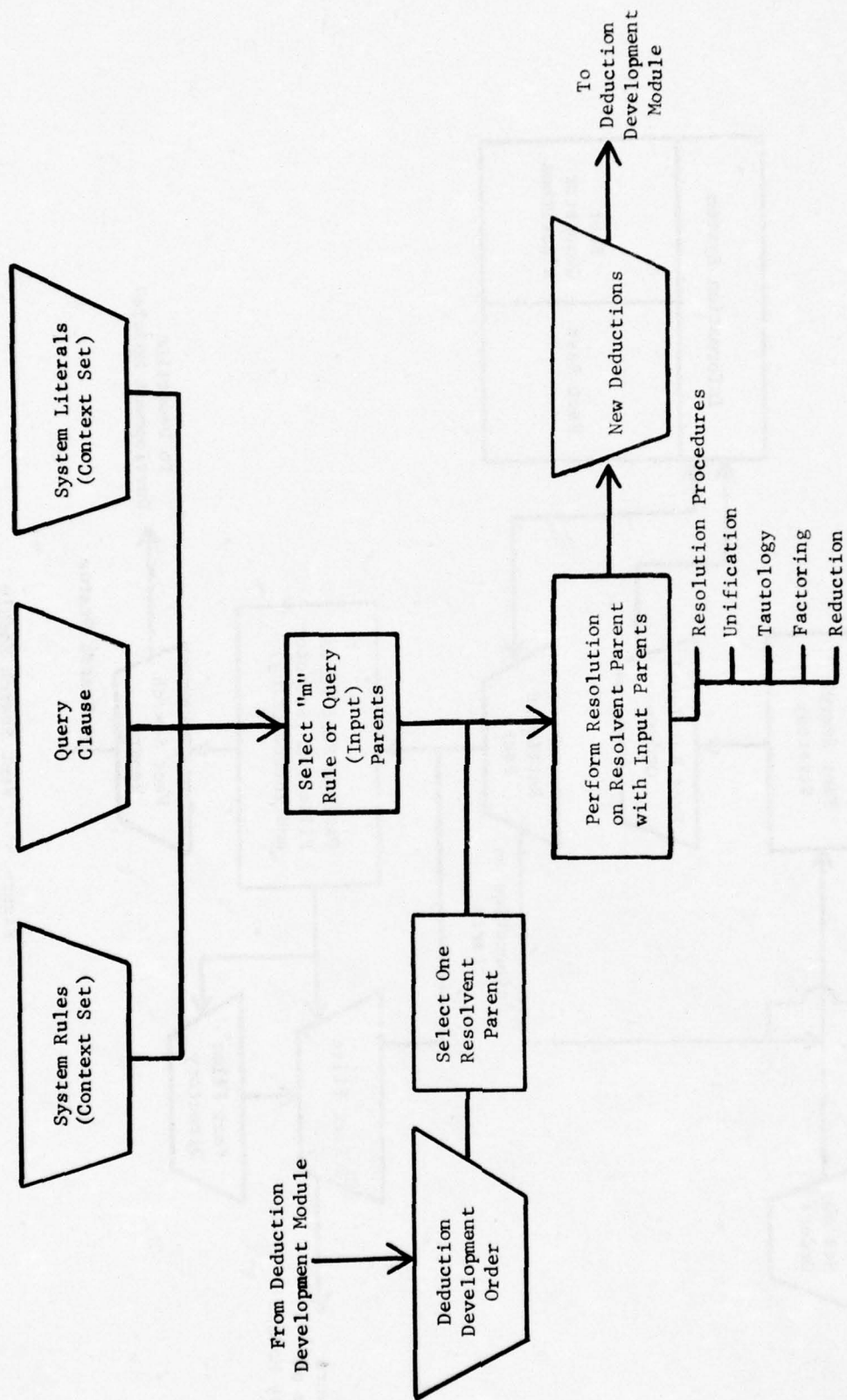


Figure 4-6 Deduction Resolution Module

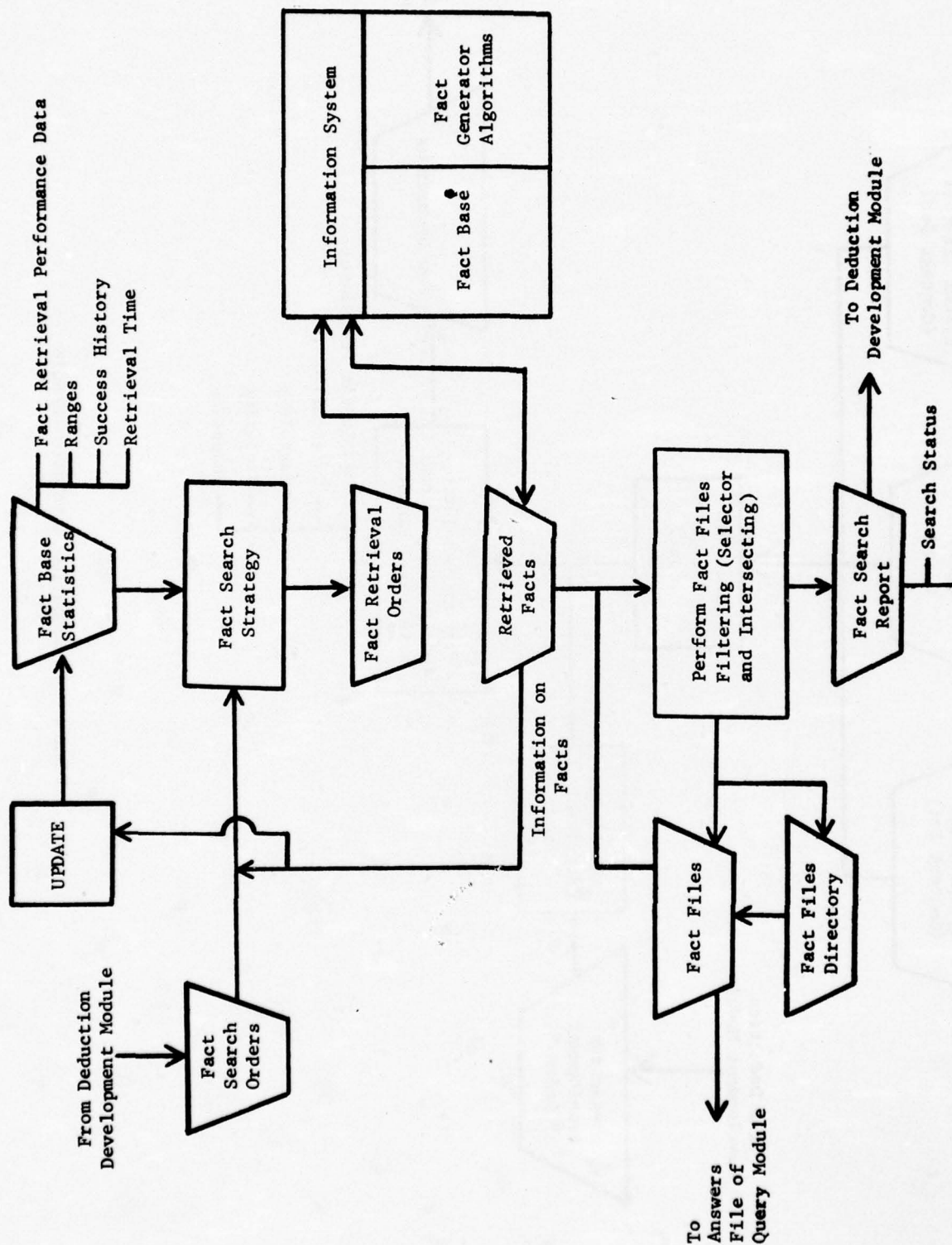


Figure 4-7 Fact Search Module

inter-dependent literals jointly. A directory is maintained by the Fact Search Module that links all retained fact files to the literals and deductions they satisfy.

The results of each fact search order are reported to the Deduction Development Module, which uses this information to recompute merit values of deductions and to delete deductions that cannot be satisfied by fact search or that have been completed. When a deduction has all of its component literals satisfied with the required quantity of fact data, fact search is completed for that deduction, and the deduction and its associated fact files are passed to the user, via the Answers file.

4.1.6 Fact Generation Algorithms

The information system provides algorithms that generate facts when presented with certain literals or literal groups (sub-clauses). Operationally, these algorithms are treated in the same manner as a fact base. The Fact Search Module issues orders to the appropriate algorithm and awaits fact files. The remaining operations are the same as described in responding to fact base files.

4.1.7 Other Modules

These include modules that perform merit performance data collection and computation, context set determination, user dialogue, and the executive module - to be developed.

4.2 DEDUCTION PROGRAM DATA DICTIONARY

This section describes tables required by the program modules of the inference system.

4.2.1 Deduction (Tree) Table

This table contains general deduction tree information and includes the node linkages of the tree (for trace purposes), active/inactive status of deduction for the tree, and summary deduction information.

TABLE 4-1
DEDUCTION TREE TABLE

A. General Information

	<u>Comment</u>
1. Tree Id	Each Query is identified
2. Query Clauses	Each query clause, in refutation form, starts a deduction tree.
3. User Input Information	User supplied merit data, etc.
4. Node i	Nodes are listed in this table in random order, since parent/child node linkages are maintained in each entry.
5. Parent Node Id of Node i	
6. Node Tree Position	
Tree level	Levels from start of tree. Sometimes called depth of tree.
Sibling position	Position with respect to other nodes of the same parent.
No. of Siblings	
7. Node Status	
Active (But not fully Resolved)	
Dead	Rejected Deduction
Completed (Fully Resolved)	To be fact determined
Special Characteristics	These designate special deduction characteristics with regard to developing new deductions
Recursive Node	
Other	
8. Deduction Fact Determination Status	Applies to partially or fully resolved deductions engaged in fact search.

B. Deduction (Node) Information

	<u>Comments</u>
1. Node Id/Tree Id	
2. Node Deduction Clause Merit Value	Used to guide in deduction development.
Probability of Success ($0 \leq X \leq 1$)	
Load Expectation	
3. Resolver Parent Id	Rule Clause or System Fact Literal Parent used to Resolve this deduction.

TABLE 4-1 (Cont'd.)

DEDUCTION TREE TABLE

C. Deduction Clause Description

1.	No. of Literals in clause	Each clause consists of a Literals list and a Term list. .
2.	List of Literals in clause	Literals are ordered right to left.
2.1	Literal ID	
2.1.1	Literal Sign	Negative or Positive Sign
2.1.2	No. of terms in Literal	
2.1.3	Literal Type	Types: Resolved; fact search in process, etc.
2.1.4	Links to terms (in Terms List below)	
2.1.4.1	Link to term 3i	
	.	
	.	
	.	
2.1.4n	Link to term 3j	
3.	Term List	This list contains the non redundant set of terms in the deduction clause. Order is not relevant
3.1	Term Id 1	
3.2	Term Id 2	
3.2	Term Id n	The terms are separated to facilitate the operation of term substitution in resolution unification operations.

4.2.2 System Rules Table

This table is similar to Deduction Tree Table, Part C, Deduction Clause Description. This is because each rule is a clause, just as a deduction is a clause. The Literal Type entry, however, may have different designations. The Rules also contain merit entries, one for each rule.

4.2.3 Deduction Development Order Table

This table contains parents sent to the Deduction Resolution Module for new deduction generation.

1. Request Order Id
 - 1.1 Deduction Id, Tree Id
 - 1.2 Status
 - (1) Process stage initiated
 - (2) Process stage completed
 - (3) No. of deductions sired, and Max. Possible No.
 - 1.3 Statistics
 - Time In
 - Time Out
2. Request Order Id

4.2.4 System Literal Table

The literals within the system are listed in this table. A sub-table is formed from this table corresponding to each particular context. This table is used by the Deduction Resolution Module to select resolver parents, once the resolver literal has been determined. It is also used to compute merits of literals. Various literal forms in the list provide different literal merit values.

The table includes variants of a literal, e.g., 1.1.3, 1.1.4,... 1.1.n in table below.

TABLE 4-2

IV. SYSTEM LITERAL LIST

1. Base Context Set Id
 - 1.1 Literal Id (Alphabetic Ordering of Literals)
 - 1.1.1 Form 1 : L (t,t,...t)
 - (1) Fact Domain of literal. Total (all sources)
 - (2) Fact Domain of literal in Fact Base Only.
 - (3) Merit (Probability of Success)
 - (4) Merit Load Expectation - Time/Cost for: Deduction Phase;
Search Phase
 - (5) No. of Rules as Resolvers
 - (6) No. of Rules containing this literal
 - 1.1.1.2 Rule Resolvers (Ordered by Rule Merit)
 - (1) Rule Id₁
|
 - (n) Rule Id_n
 - 1.1.1.2 Form 2 : L(-,t,...,t)
 - 1.1.1.3 Form 3 : L(t,-,t,lll,t)
 - 1.1.1.N Form N : L(-,-,...,-)
 - 2 Base Context Set Id
 - 1.2 Literal Id
 - |
 - |
 - |

4.2.5 Deduction Merit Ranking Table (Active Nodes Only)

This Table contains: List of Tree node Deduction Id's ordered by Merit Probability of Success.

1.	Node Id11	Highest Merit
2.	Node Id12	Next highest Merit
N.	Node Idin	Least Merit

4.2.6 Deduction Fact File Base (DFFB)

This is a file collection of facts acquired from the Fact Base and Fact Generators (computer programs). These files are maintained by the inference for fact determination processing (e.g., fact intersecting) and for providing answers to users on their queries.

This DFFB contains a directory of its files linked to active deductions which are, and have been, fact searched. The following information is maintained for each directory entry.

(DFFB) Directory Entries

1. Deduction Id
2. Resolved Literals Fact Searched
 - 2.1 Literal Id
 - 2.1.1 No. of Answers
 - 2.1.2 DFFB location of Facts
 - 2.1.3 Status of Search
 - 2.1.4 Retrieval time interval
 - 2.2 Literal Id2
 - 2.n Literal Idn
3. Summary Search Information

4.2.7 Performance Information Table (Statistics)

This table contains performance information on literals in relation to a queries processing.

1. Literal Id
 - 1.1 Resolved by (Rule Id)
 - 1.2 Average time for literal verification (full fact retrieved)
 - 1.3 Size of deduction tree when literal was satisfied
 - 1.4 Average no. of facts retrieved for literal
 - 1.5 Average of facts satisfied deduction
2. Literal Id2

4.3 SOME PROGRAMMING DESIGN TECHNIQUES

4.3.1 Resolution Unification Mechanism

This section presents a table structured mechanism for resolving deductions (an optimized algorithm is presented in Appendix E).

Two parent clauses generate a deduction by resolving on a common literal. The common literal is in positive form in one parent and is in negative form in the other parent. Prior to resolution, the terms of the resolving literals in the two parents may differ in their domains. A process of unification is applied which finds a common domain, generally the greatest common domain for the terms of the two resolving literals. Following this, a proper substitution of terms in the non-resolving literals of the parents is applied to maintain term consistency. This is illustrated by the following example:

Parent 1: $L_1(X) L_2(X,Y) L_3(1)$

Parent 2: $L_4(X,Z) \bar{L}_3(Y)$

Resolving Literals are: $L_3(1)$ and $\bar{L}_3(Y)$

Greatest Common Domain forces $Y = 1$. This is unification.

Unified Parents are:

Parent 1 $L_1(X) L_2(X,1) L_3(1)$ Includes substitution into non-resolving literals.

Parent 2 $L_4(X,Z) \bar{L}_3(1)$

Resolved Deduction

$L_1(X) L_2(X,1) \boxed{L_3(1)} L_4(X,Z)$

The framed literal is the resolvent literal.

Table 4-3 illustrates how the resolution procedure is mechanized for the following example.

Rule Parent : $L_4(Z,B,X) L_3(C,Y,Z) L_2(Y,A,Z) \bar{L}_1(Y,Z,X)$

Deduction Parent : $L_6(X,A,Z) L_5(C,Y,Z) L_4(Z,Y,C)$

Resolvent Literals : $\bar{L}_4(Z,B,X)$ and $L_4(Z,Y,C)$, also $B \subset Y$ and $C \subset X$

Unification Substitution $Y \rightarrow B, X \rightarrow C$

Child : $L_6(C,A,Z) L_5(C,B,Z) \boxed{L_4(Z,B,C)} L_3(C,B,Z) L_2(B,A,Z) L_1(B,Z,C)$

4.3.2 Factoring Mechanization

Factoring is a resolution operation which can be applied to eliminate a literal in a clause when it appears repeatedly in the clause. For example, $L_1(X) L_2(Y) L_1(X)$ factors to $L_1(X) L_2(Y)$. Care must be exercised to note that factoring has been applied in the resulting clause; this is important to later possible operations on the clause.

TABLE 4-3

RESOLUTION UNIFICATION TABULAR EXAMPLE

Rule Clause	Parent Deduction Clause
Literals of Clause $S_4L_4, \text{Type}, \text{TTL}_1, \text{TTL}_2, \text{TTL}_3$ $S_3L_3, \text{Type}, \text{TTL}_4, \text{TTL}_5, \text{TTL}_1$ $S_2L_2, \text{Type}, \text{TTL}_5, \text{TTL}_6, \text{TTL}_1$ $S_1L_1, \text{Type}, \text{TTL}_5, \text{TTL}_1, \text{TTL}_3$	Literals of Clause $S_6L_6, \text{Type}, \text{TTL}_1, \text{TTL}_2, \text{TTL}_3$ $S_5L_5, \text{Type}, \text{TTL}_4, \text{TTL}_5, \text{TTL}_3$ $S_4L_4, \text{Type}, \text{TTL}_3, \text{TTL}_5, \text{TTL}_4$
Terms of Clause TT1: Variable Z TT2: Constant B TT3: Variable X TT4: Constant C TT5: Variable Y TT6: Constant A	Terms of Clause TT1: Variable X TT2: Constant A TT3: Variable Z TT4: Constant C TT5: Variable Y
Child Deduction Clause Literals of Clause $S_6L_6, \text{Type}, \text{TTL}_4, \text{TTL}_2, \text{TTL}_3$ $S_5L_5, \text{Type}, \text{TTL}_4, \text{TTL}_1, \text{TTL}_3$ $S_4L_4, \text{Type}=\square, \text{TTL}_3, \text{TTL}_1, \text{TTL}_4$ $S_3L_3, \text{type}, \text{TTL}_4, \text{TTL}_1, \text{TTL}_3$ $S_2L_2, \text{type}, \text{TTL}_1, \text{TTL}_2, \text{TTL}_3$ $S_1L_1, \text{type}, \text{TTL}_1, \text{TTL}_3, \text{TTL}_4$	
Terms of Clause TT1: Constant B TT2: Constant A TT3: Variable Z TT4: Constant C	
Symbols:	Sn = Sign of Literal Ln Li = ith Literal in clause Type = Designates character of the Literal, e.g. \square means a resolvent Literal TTi = A term of a Literal in the clause TTLi = A link to the term of a Literal in the clause

Table 4-4 illustrates how this factoring is handled using the tabular structure for clauses. Actually, only one simple change takes place that is the "type" of the factored literal is changed to designate that the literal has been factored.

Note that a unification operation can lead to a factoring operation. For example, given $A \subset Y$, $L_1(X,Y)L_2(Y,Z)L_1(X,A)$ can be unified to generate a new deduction $L_1(X,A)L_2(A,Z)L_1(X,A)$ which can be factored to $L_1(X,A)L_2(A,Z)$.

TABLE 4-4

FACTORING OPERATION

Deduction Clause = $L_1(X,A)L_2(A,Z)L_3(Y,A)L_2(A,Z)$

Literal List	Literal Description
$S_2L_2, \text{Type}, TTL_3, TTL_4$	$L_2(A,Z)$
$S_3L_3, \text{Type}, TTL_2, TTL_3$	$L_3(Y,A)$
$S_2L_2, \text{Type}, TTL_3, TTL_4$	$L_2(A,Z)$
$S_1L_1, \text{Type}, TTL_1, TTL_3$	$L_1(X,A)$

Term List

TT1: Variable X
 TT2: Variable Y
 TT3: Constant A
 TT4: Variable Z

Factored Deduction Clause = $L_1(X,A)L_2(A,Z)L_3(Y,A)$

Literal List

$S_2L_2, \text{Type} = \text{Factor}, TTL_3, TTL_4$
$S_3L_3, \text{Type}, TTL_2, TTL_3$
$S_2L_2, \text{Type}, TTL_3, TTL_4$
$S_1L_1, \text{Type}, TTL_1, TTL_3$

Term List

TT1: Variable X
 TT2: Variable Y
 TT3: Constant A
 TT4: Variable Z

APPENDIX A

REAL AND INFERRED ANSWERS TO QUERIES

REAL AND INFERRED ANSWERS

A. Types of Deduction Answers

The inference system makes available to the inquirer three types of answers, in the form of deduction (disjunctive) clauses. They are, in the framed notation of this report:

I. Complete and Real Deductions

Example Deduction:

A _{xy}	B _{yz}	C _r	S _{rt}
-----------------	-----------------	----------------	-----------------

The frames mean that all literals have been resolved by system inference rules, and the slash in the frame, i.e.,

--

 means that the desired quantity of fact data has been retrieved for each literal, either from the fact base or from a computer fact generator.

II. Complete and Inferred Deduction

Example Deduction:

A _{xy}	B _{yz}	C _r	S _{rt}
-----------------	-----------------	----------------	-----------------

This means that all literals have been resolved by system inference rules, but that for literals not in a slashed frame, the quantity of desired facts have not been acquired from the fact base or fact generators. As a result the deduction is not verified by facts, rather it is affirmed by the system inference rules. If a user analyst accepts such a deduction, he in effect infers the completion of the deduction.

III. Incomplete Deductions

Example Deduction

A _{xy}	B _{yz}	C _r	S _{rt}
-----------------	-----------------	----------------	-----------------

This means that not all literals of the deduction have been resolved by system rules.

B. Premises of an Inferred Literal

A literal in a deduction may be interdependent with other literals in the deduction if the domains of their terms overlap. For example in the deduction: $A_{xy} B_x C_r D_{rs} E_t$, A_{xy} and B_x are interdependent, but independent of the others. In connection with this, the literals upon which an inferred literal is dependent are called its premises. In the case of an Inferred Deduction, each inferred literal may have real or other inferred literals as its premises. The concept of premises will have meaning to the user analyst, and also to the deduction mechanism.

C. Mechanization of Premises

Premise pointers are used to mechanize deduction derivation. Stored with each inferred-framed literal are pointers to its premises. As a literal becomes fact satisfied, premise pointers to it are dropped. When a literal no longer has any premise pointers it is recognized as fully satisfied.

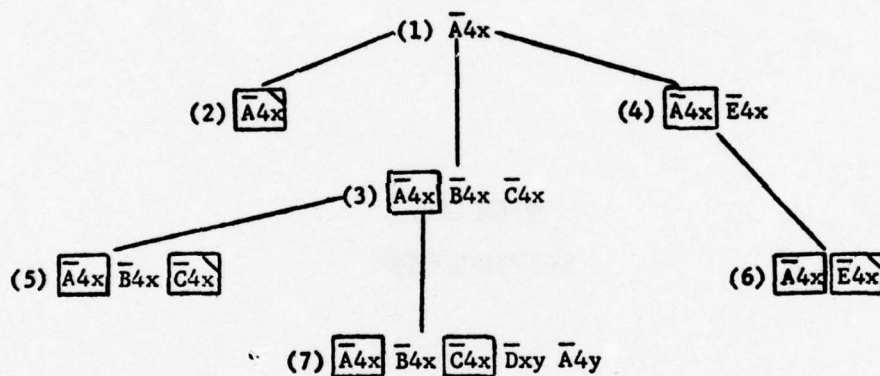
APPENDIX B
RECURSIVE NODES

A. GENERAL

In the OL-deduction procedure, a parent deduction (tree node) is chosen and its rightmost literal is resolved by a parent node or query clause to form a new deduction.

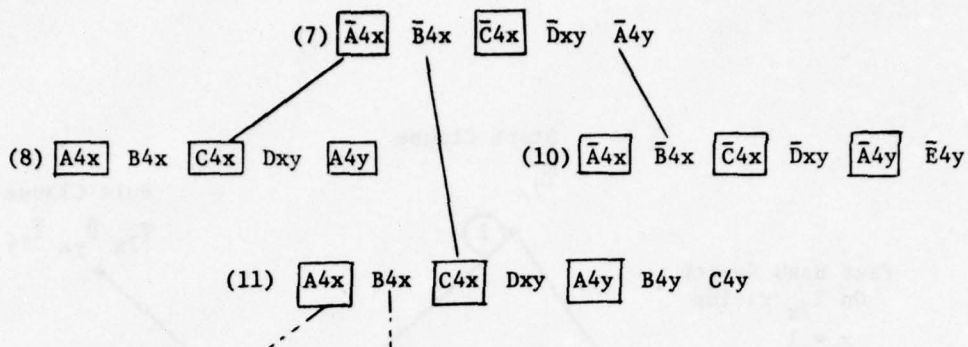
In seeking to satisfy a literal, a satisfying of the same literal or a more general literal may already have been started in an ancestor deduction higher up in the deduction tree.

For example: (Rule parents are not shown)



In this deduction tree, we see that solutions (i.e., facts, derived from a fact base) that satisfy the rightmost literal of node 1 are also solutions to the rightmost literal of node 7. Any such solution to $\bar{A}4y$ that we can derive from node 7 is better derived under node 1; and it is useless to rederive such answers to $\bar{A}4y$ at node 7. There is however a purpose in expanding the tree at node 7, as a generator of answers.

Expanding the deduction tree at node 7 generates a duplication of the parent tree starting at node 1. This is illustrated by:

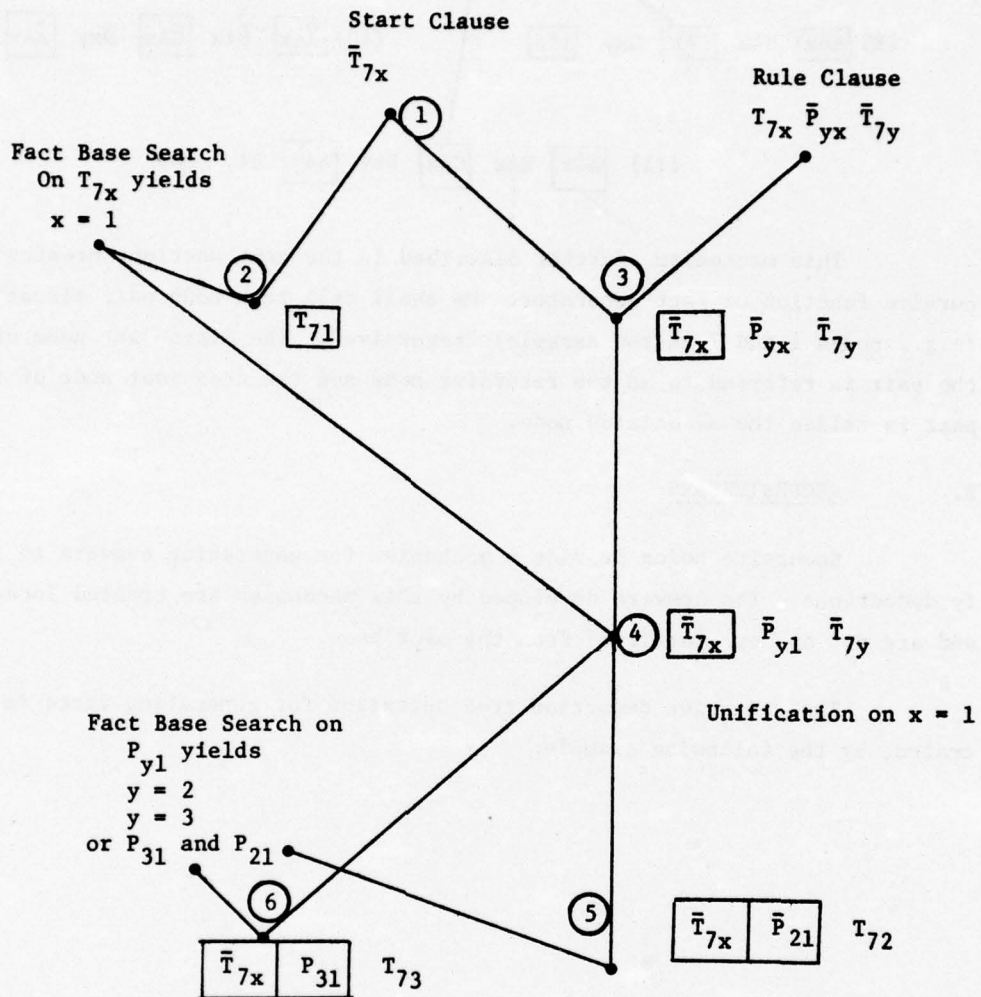


This mechanism, further described in the next section, creates a recursive function or fact generator. We shall call such node pair situations (e.g., nodes 1 and 7 in the example) "recursive". The descendant node of the pair is referred to as the recursive node and the tree root node of the pair is called the associated node.

B. RECURSIVENESS

Recursive nodes provide a mechanism for generating answers to satisfy deductions. The answers developed by this mechanism are created locally and are not answers retrieved from the fact base.

The recursive deduction tree operation for generating facts is illustrated, by the following example:



T_{72} and T_{73} are answers (inferred) that are generated by this procedure. T_{71} , P_{21} , and P_{31} , however, were fact base derived answers.

Node 3 is recursive with associated node 1, and can be resolved with one or more answers already known to satisfy node 1. Suppose, now, node 2, via a fact base search, acquires an answer $x = 1$, and therefore T_{71} . This answer automatically also satisfies node 1. The value $x = 1$ can also be instantiated (a process implying unification) into node 3 such that P_{yx} becomes P_{y1} in node 4. Upon a further fact base search, let us assume as returns for P_{y1} , the satisfying values $x = \{2,3\}$. With further instantiation of these values with node 4 we derive two new answers, T_{72} and T_{73} in nodes 5 and 6 respectively. This process can continue recursively until no more facts are found, or we reach the maximum numbers of desired answers.

Note: Recursive node answer generation is generally started by at least one fact base answer.

APPENDIX C
PARTIALLY TRANSITIVE RULES
AND RECURSIVE DEDUCTION TREES

Almost all rules or classes of rules, when analyzed in depth, will display characteristics that can be used to guide deduction tree development. This section presents an analysis of one of these rule classes generally described as a "partially transitive rule" class.

A. PARTIALLY TRANSITIVE RULES

A partially transitive rule has the disjunction clausal form:

$$\bar{B}_{xy} \bar{A}_{xz} A_{yz}, \text{ or the implicative form: } B_{xy} \wedge A_{xy} \Rightarrow A_{yz}.$$

A partially transitive rule can also be expressed by the form P (Ar, Bs), where

A = the shared relation of x and y to z

B = the relation between the terms x and y

r = specifies the (1 or 2) position of term z in A

s = specifies the (1 or 2) position of term y in B

Thus,

$$(1) \quad P(A_2, B_2) \quad \text{or} \quad B_{xy} \wedge A_{xz} \Rightarrow A_{yz}$$

$$(2) \quad P(A_1, B_2) \quad \text{or} \quad B_{xy} \wedge A_{zx} \Rightarrow A_{zy}$$

$$(3) \quad P(A_2, B_1) \quad \text{or} \quad B_{yx} \wedge A_{xz} \Rightarrow A_{yz}$$

$$(4) \quad P(A_1, B_1) \quad \text{or} \quad B_{yx} \wedge A_{zx} \Rightarrow A_{zy}$$

Notes:

(1) When B is symmetric, the subscript is dropped since rules 1 and 3 become identical as do rules 2 and 4.

(2) The pure transitive rule $P(A_r, A_s)$ is a special case of the partially transitive rule.

A partially transitive rule is apt to produce a recursive deduction tree. For example, application of the rule $P(X1, Z2) = X_{zy} \bar{Z}_{xy} \bar{X}_{zx}$ to the literal \bar{X}_{3y} produces the recursive node

$$\boxed{\bar{X}_{3y}} \bar{Z}_{xy} \bar{X}_{3x} \quad \text{where } x \text{ and } y \text{ are equally general}$$

This node while redundant for fact base searches can, however, be applied as a recursive answer generator. For example, given a single value of y, it becomes possible to generate satisfying values of x; these can then

be used recursively when the domain of x and y overlap. That is, to y may be applied the new values of x , to further generate new values of x . This recursive procedure stops when the values of x that are generated are not new; that is, when the values of x that are generated have previously been generated.

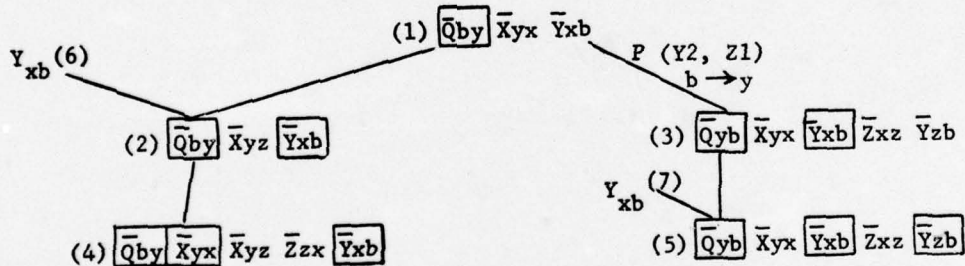
C. REDUNDANT DEDUCTIONS: A SPECIAL CASE

When two literals in the same node may be resolved by a similar partially transitive rule, a great deal of redundancy may result. Two partially transitive rules are similar when the rules have the forms $P(A_r, B_s)$ and $P(C_r, B_w)$, respectively.

Example: Similar Partially Transitive Rules

$$\begin{aligned} P(X1, Z2) &= \bar{Z}_{zx} \bar{X}_{yz} X_{yx} \\ P(Y2, Z1) &= \bar{Z}_{xz} \bar{Y}_{zy} Y_{xy} \end{aligned}$$

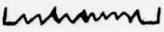

and Y_{xb}



In this example, the literals X and Y each have a partially transitive rule with the literal Z , they share a term, and they are both members of the same deduction clause (see top node). This structure allows the development of a pair of nodes (4 and 5) whose remaining unresolved literals are virtually the same. Also, the resolved literals requiring fact base retrievals are the same. Consequently, the work required on both deductions nodes (4 and 5) is the same. In such situations we should restrict use of both partially transitive rules in deduction operation

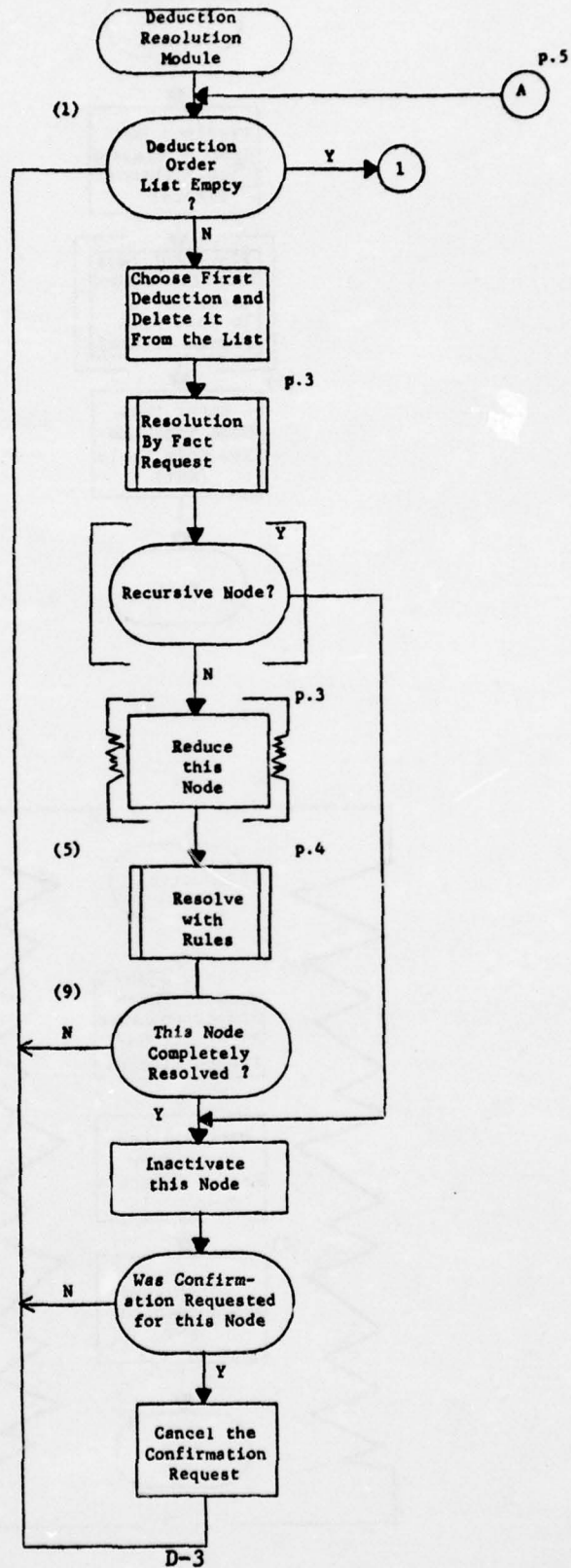
APPENDIX D
DEDUCTION RESOLUTION MODULE-PROCEDURAL FLOW CHARTS

This Appendix presents procedural flow charts for the deduction resolution module.

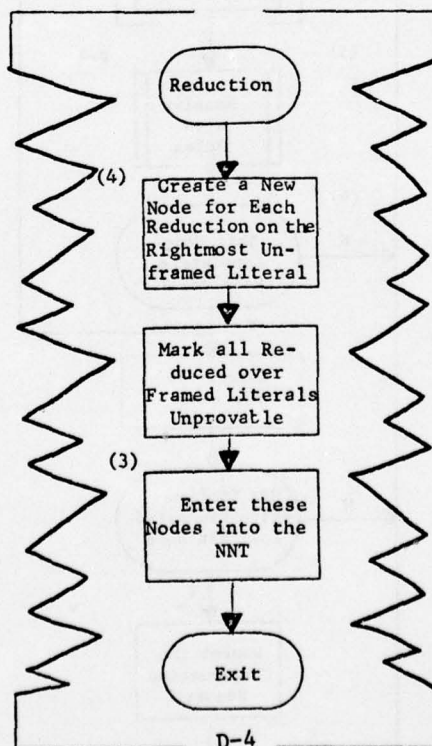
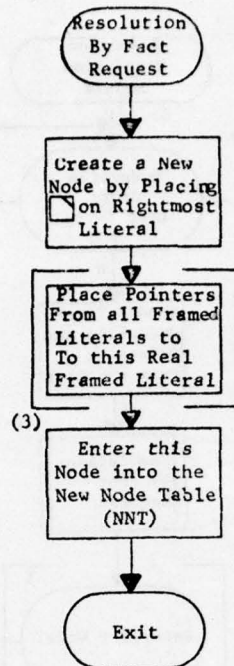
Sections of the flow charts not applicable to SNL are denoted by the following marking:  Sections involved with advanced concepts not needed for a minimum system are denoted by: 

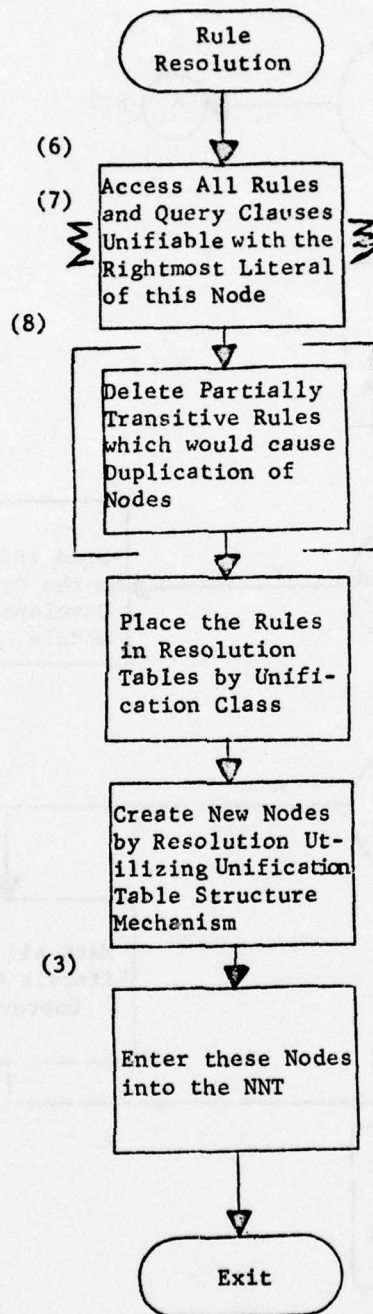
Numbers in parentheses within the flow charts refer to notes found at the end of this Appendix.

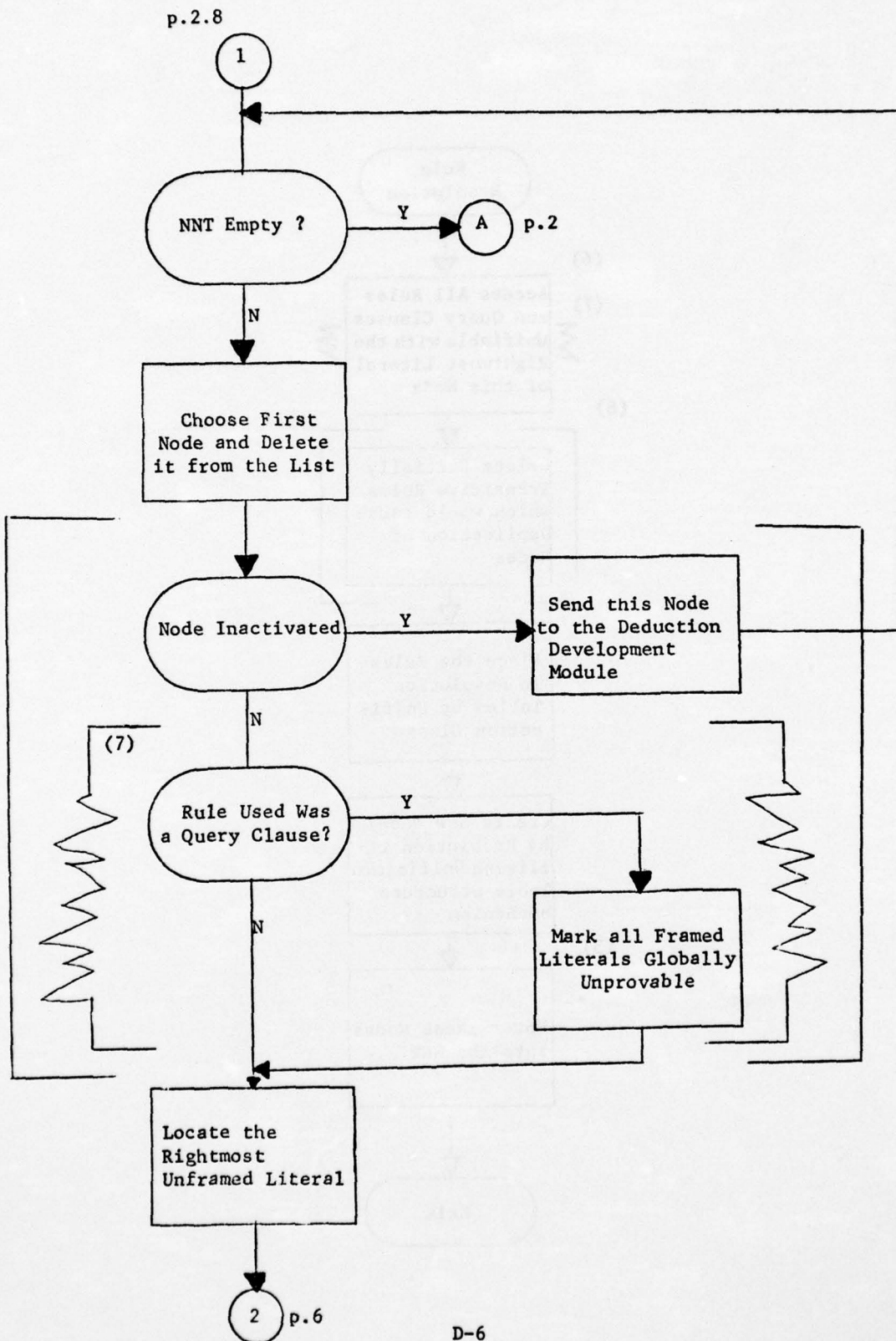
Certain concepts referenced in the flow charts such as "recursive node" are explained in the other Appendices.

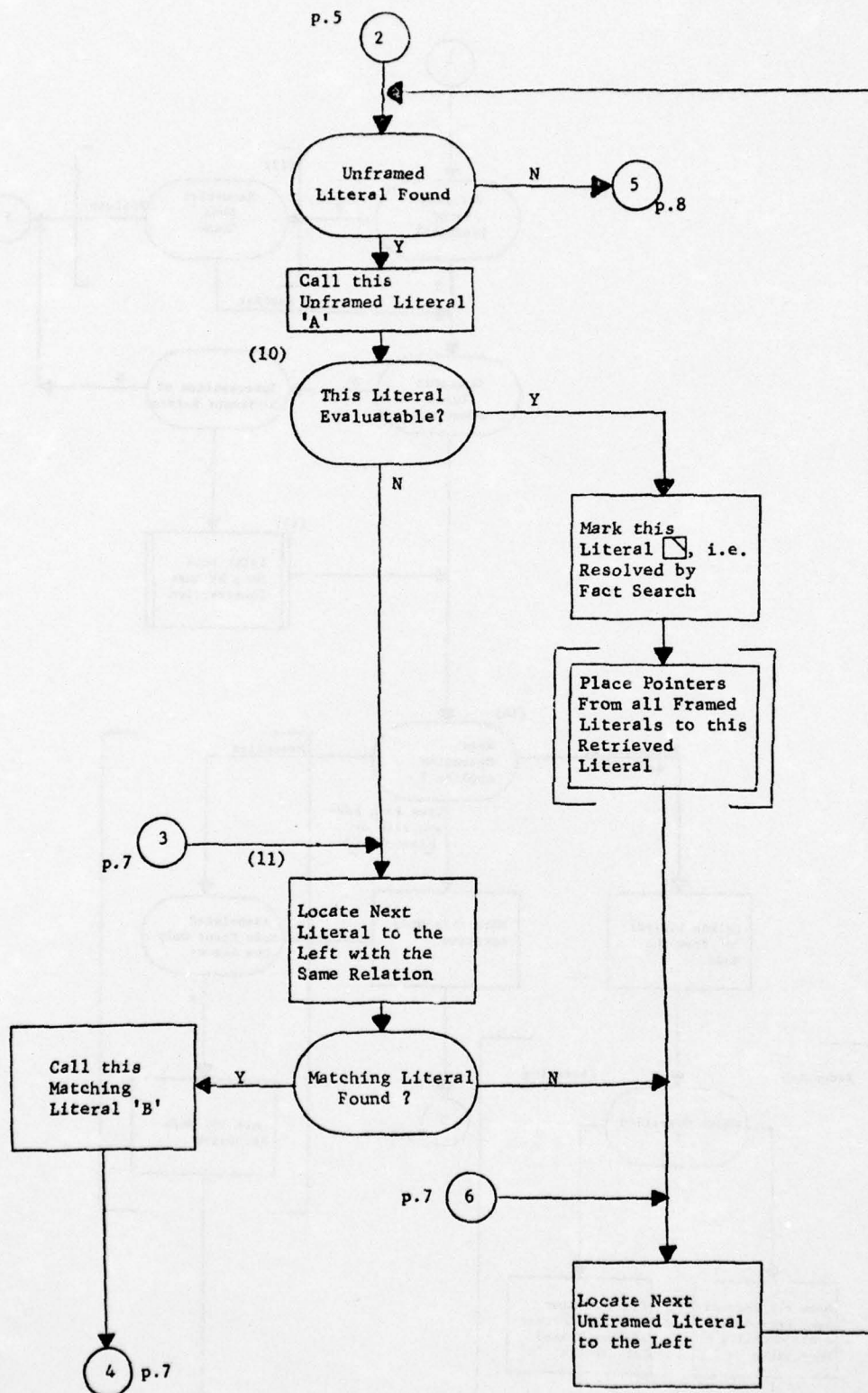


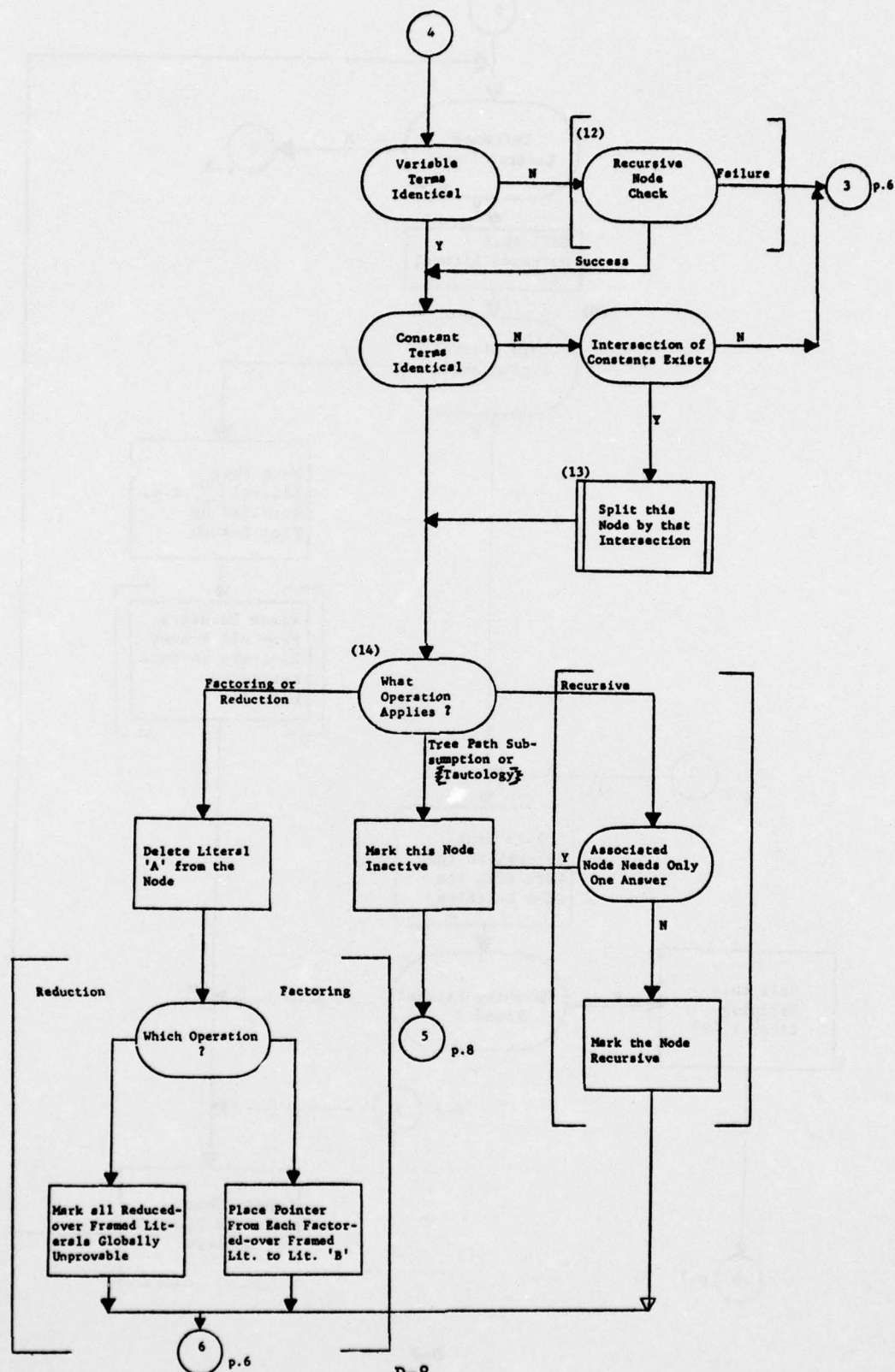
(2)

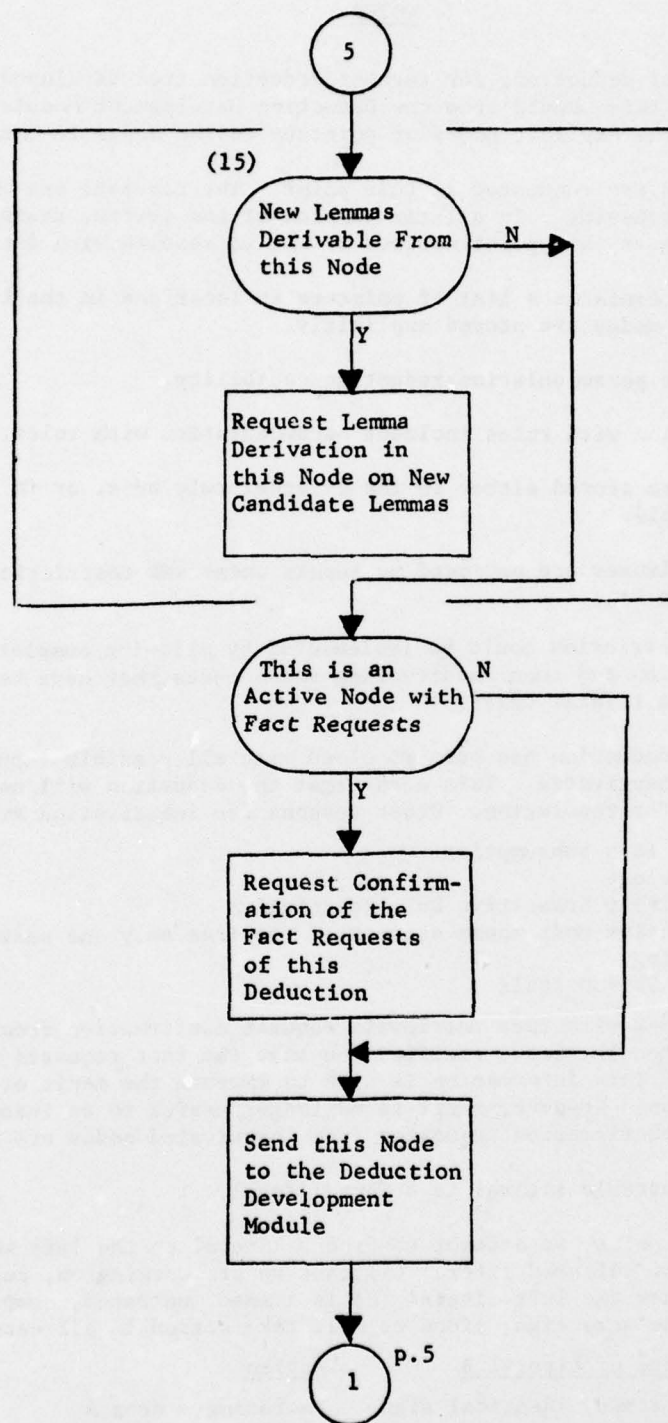












NOTES

1. A list of deductions for further deduction tree development has been sent to this module from the Deduction Development Module. The list may be the explicit nodes or pointers to the nodes in the tree.
2. No facts are requested at this point. The requests are made after post-processing. In a later version of the system, there may be a decision at this point whether or not to resolve with facts.
3. The NNT contains a list of pointers to locations in the tree where the new nodes are stored explicitly.
4. Includes paramodulation-reduction capability.
5. Resolution with rules includes paramodulation with rules.
6. Rules are stored either in the external rule base, or in a local rule table.
7. Query clauses are not used as inputs under SNL restrictions. (See Appendix F)
8. This restriction could be implemented by allowing complete rule resolution and then inactivating those nodes that have been created with the illegal rules.
9. Once a deduction has been resolved with all possible input clauses, it is inactivated. This means that the deduction will never be chosen for resolution. Other reasons for inactivation are:
 - Tree Path Subsumption
 - Tautology
 - Partially Transitive Rule Restriction
 - Recursive node whose associated requires only one answer
 - Pruning
 - Fact Search failsDeductions with fact retrievals request confirmation from the Fact Search Module, i.e., verification that the fact requests are satisfiable. This information is used to improve the merit of the deduction. However, merit is no longer useful to an inactive node, so any confirmation requestes from inactivated nodes are cancelled.
10. An evaluable literal is a fact literal.
11. At this point, we attempt to find a literal to the left which will match the unframed literal (A) that we are working on, regardless of whether the left literal (B) is framed, unframed, complementary, or of the same sign; since we will take action in all cases.

State of Literal B

Action

Unframed, identical sign
Unframed, complement
Framed, identical sign
Unframed, complement

Factoring - drop A
Tautology - inactivate the node
Chain Subs. - inactivate the node
Reduction - drop A

12. All post-processing operations (factoring, reduction, tree path subsumption, tautology) require that terms of 'A' and 'B' are identical. If variable terms are different, then the only remaining possibility of interest is that the new node may be recursive. (See Appendix E)

To be absolutely sure of a possible recursive node, the checks should be made between the unframed literal A and the rightmost literal of the root node of literal B.

13. Splitting is a procedure by which a node which contains multiple instantiations is separated into two nodes. These 2 nodes are identical to the first except for one or more constant terms. The sum of the new nodes equals the old node.

The new nodes are created by altering the old node and creating one new one. The splitting is done by the intersection of one constant list in the node with another list (constants in a rule or fact or in a previous literal in the node). The new node receives the intersecting terms while the old node keeps only the non-intersecting terms.

Example 1:

$$\boxed{\bar{A} \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} \begin{pmatrix} 5 \\ 7 \\ 9 \end{pmatrix}} \bar{D} \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} \times \boxed{\bar{C}x \begin{pmatrix} 5 \\ 7 \\ 9 \end{pmatrix}} \bar{D} \begin{pmatrix} 1 \\ 2 \\ 5 \end{pmatrix} \times$$

This node can be split by the intersection of $\begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$ and $\begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$ into

New Node: $\boxed{\bar{A} \begin{pmatrix} 1 \\ 2 \\ 7 \end{pmatrix} \begin{pmatrix} 5 \\ 7 \end{pmatrix}} \bar{D} \begin{pmatrix} 1 \\ 2 \end{pmatrix} \times \boxed{\bar{C}x \begin{pmatrix} 5 \\ 7 \end{pmatrix}} \bar{D} \begin{pmatrix} 1 \\ 2 \end{pmatrix} \times$ and

Old Node: $\boxed{\bar{A}39} \bar{D}3x \boxed{\bar{C}x9} \bar{D}5x$

The first node is factored into

$$\boxed{\bar{A} \begin{pmatrix} 1 \\ 2 \\ 7 \end{pmatrix} \begin{pmatrix} 5 \\ 7 \end{pmatrix}} \bar{D} \begin{pmatrix} 1 \\ 2 \end{pmatrix} \times \boxed{\bar{C}x \begin{pmatrix} 5 \\ 7 \end{pmatrix}}$$

Example 2:

Sometimes a constant is not yet defined due to pending fact lockup. Splitting may still be done. e.g.:

$\boxed{\bar{B}1x} \boxed{\bar{A}1x} \bar{B}cx \boxed{\bar{D}1c}$ is split into

New Node $\boxed{\bar{B}1x} \boxed{\bar{A}1x} \bar{B}1x \boxed{\bar{D}11}$ and Old Node $\boxed{\bar{B}1x} \boxed{\bar{A}1x} \bar{B}cx \boxed{\bar{D}1c} c \neq 1$

The first node is immediately inactivated by tree path subsumption.

In this procedure, the old node is placed at the top of the NNT while the new node continues post-processing.

14. Which operation applies depends on the framing and sign of literal B as described in note 11.
15. A node is useful in the derivation of new lemmas if:
- (1) A new real framed literal was added to this node, or
 - (2) There is an inferred-framed literal which, because of the last resolution, now meets the following criteria:
 - a. No unframed literals to the right of it,
 - b. All premise literals satisfy criteria a and b.

This prevents the derivation of the same lemma more than once in the same path.

APPENDIX E

OPTIMIZED PROGRAM DESIGN FOR UNIFICATION - RESOLUTION PROCEDURE

This section describes a design for optimizing the program that implements the Unification-Resolution Procedure. The design incorporates a table structure for representing clauses in the unification resolution process that consolidates common entries. While this design is more complex than the design presented in the main body of this report, it offers a reduction in storage and an improvement in processing speed

The design presented is an adaptation of the method described in "Computational Logic: The Unification Computation" by J. A. Robinson. A simpler, non-optimal design, is presented in the main text, section 4.3.1.

A. NODE REPRESENTATION

All nodes of a deduction tree are defined by four tables:

- (1) Constant Table (CT),
- (2) Node Table (NT),
- (3) Literal Table (LT),
- (4) Term Table (TT).

A.1 Constant Table (CT)

The CT contains all constants that are instances of terms appearing in the literals of nodes in the deduction tree. The entries in the CT are linked by pointers to these related terms in the Term Table (TT). Entries may be single constants or arrays.

A.2 Node Table (NT)

The NT represents all nodes of the deduction tree. Each row of the NT represents one node. Nodes enter the NT and are arranged in the table in their order of creation. For each node, the table contains the following information:

- (0) ID number
- (1) Number of literals in this node
- (2) A list of pointers that link to entries in the Literal Table (LT). This is an ordered list which denotes the left-to-right order of the literals in this node.

For each pointer the following associated information is provided:

- The framed or unframed status of this literal
 - The fact retrieved status of the literal (being looked up in the fact base)
 - Links to premise literals
- (3) A pointer to a term table segment (which defines the terms of the literals of this node).
- (4) Other information about the node:
- Node Merit
 - Active Status
 - Pointer to Resolvent Parent Node (Resolvent Clause or Top Clause of Deduction Tree)
 - Pointer to Input Parent Node

A.3 Literal Table (LT)

LT represents the literals of the nodes of the deduction tree. Each row of the LT represents a specific literal. Whenever new literals (e.g. via rules) enter the deduction tree development they are added to the LT. LT contains the following information:

- (0) ID number (implicit or explicit)
- (1) Literal ID
- (2) Sign - positive or negative
- (3) Arity - number of terms
- (4) List of pointers that link to the literals terms which are maintained as entries in segments of the term table.

Note: The literal table can be augmented but not updated, i.e., once a row is created, it is not modified.

A.4 Term Table

The terms of the literals of the nodes of a tree are defined by a segmented term table. Each node links to a segment. Each segment is sufficient to define the terms of one or more nodes.

Each row in the TT represents a term. The following information is recorded for each term:

AD-A031 481

AUERBACH ASSOCIATES INC PHILADELPHIA PA
A DEDUCTIVE SYSTEM FOR INTELLIGENCE ANALYSIS.(U)
JUL 76 I. L. GOLDBIRSH, R CARSON

F/G 9/2

UNCLASSIFIED

RADC-TR-76-199

F30602-74-C-0250
NL

2 OF 2
ADA031481



END

DATE
FILMED
12 - 76

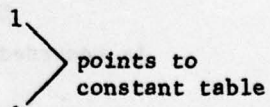
- (0) ID number (implicit or explicit)
- (1) Term type - variable (V) or constant (C)
- (2) Substitution value
 - If the term is a constant, this field contains a pointer to its value in the constant table.
 - If the term is a variable, and has not been unified, this field is empty.
 - If the term is a variable and has been unified to another term, this field contains a pointer to that other term. (Unified terms are in the same segment.)

A TT segment is created for each node. Once created, it is not modified.

A.5 Example: The nodes (1): $\bar{A}xy$ $\boxed{\bar{B}4x}$ $\bar{C}4x$ and (2): $\bar{A}x3$ can be represented as follows:

<u>NT</u>			
<u>ID Number</u>	<u>Number of Literals</u>	<u>Pointers to Literal Table</u>	<u>Pointers to Term Table Segment</u>
1	3	1, $\boxed{2}$, 3	I
2	1	1	II

<u>LT</u>				
<u>ID Number</u>	<u>Relation ID</u>	<u>Sign</u>	<u>Arity</u>	<u>Pointers to TT</u>
1	A	-	2	1 2
2	B	-	2	3 1
3 1	C	-	2	3 1

<u>TT</u>			
<u>Segment</u>	<u>Term ID</u>	<u>Term Type</u>	<u>Subst Value</u>
I	1	V	-
	2	V	-
	3	C	1
II	1	V	
	2	C	

<u>CT</u>	
<u>ID Number</u>	<u>Value</u>
1	4
2	3

Looking solely at the term table pointers in the first entry in the literal table (A), it cannot be determined which segment of the term table these apply to. This determination must be traced through the NT. The purpose of this structure is to facilitate passing essentially unchanged literal relations from parent node to descendant node with but at most a change in terms. Also, when the passed-on literal is not changed, even to its term, it need not be duplicated in the tables.

The example in the following portions of this report are based on the clauses represented above.

B. RESOLUTION BY FACT REQUESTS

When a deduction is resolved, as a parent, to create one or more new nodes, at least one new node term table segment is created, a new entry is added in the node table for each new node, and an entry is added in the literal table for each literal added by rule resolution.

In fact resolution, no new literals are added so the literal table remains unchanged. No matter how many results are found in the fact base, all the information can be expressed with one new node entry and one new term table segment.

B(1) Mechanism

- (1) Duplicate the term table segment of the parent.
- (2) Add a new term into the TT for each instantiable term. Designate its term type "Constant". Set the subst value of the new term to point to the constants location in the CT. This location will probably not be known at the time of creation of this node. However, it will eventually be supplied by the fact search module.
- (3) Set the subst value of the variable term being instantiated to point to the new constant term.
- (4) Create a new entry in the NT for this new node.

Example:

Suppose we decide to resolve node #1 with facts. The following procedure is followed:

- (1) Create segment III of the Term Table as a duplicate of segment I of node 1.
III 1. V -
 2. V -
 3. C 1
- (2) The rightmost literal is $\bar{C}4x$. There is one instantiable term x so add one constant term with presently unknown pointer.
III 1. V
 2. V
 3. C 1
 4. C -

- (3) Set subst. value of the instantiable term, term #1, to point to the new constant term, term #4.

III	1.	V	4
	2.	V	
	3.	C	1
	4.	C	-

- (4) Add a new node to the node table.

				<u>NT</u>	
ID#	3.	3	1,	<u>2</u> ,	<u>3</u> III

The frame around the 3 denotes that literal #3 is being fact searched. If there are no answers, this node is inactivated. If answers exist, the FSM stores the location of these answer(s) in the CT. At this point, the literal 3 may be dropped.

The final alterations are:

				<u>TT</u>
III	1.	V	4	
	2.	V		
	3.	C	1	
	4.	C	3	

				<u>NT</u>	
3.	2	1,	<u>2</u>	III	

B(2) Resolution With a Set of Rules

When resolving with facts, an entire set of related facts could resolve with a node at once using only one term table segment and only one entry in the node table. When resolving with rules, a related group of rules can also resolve together using one term segment but producing different entries in the node table.

B(3) Unification Classes

A group of rules may resolve simultaneously if they are of the same unification class, i.e., their resolving literal is the same. When a node is chosen for rule resolution, a number of applicable unification classes are accessed.

A literal table, node table, and a term table with one segment are developed for these rules. There is one set of tables for each unification class. For example, suppose node 2 was chosen for rule resolution.

Node 2 = $\bar{A}x3$

A request would be made for all rules in all unification classes which can resolve with this literal. Here are some sample rules:

Axy $\bar{A}xz$ $\bar{B}yz$	}	Unification Class #1 Axy
Axy $\bar{A}xz$ $\bar{C}zy$		
Axy $\bar{D}xz$ $\bar{E}zy$		
Ax3 $\bar{D}x3$	}	Unification Class #2 Ax3
Ax3 $\bar{G}3x$		

These rules would be placed in the following structures:

\underline{NT}^1	\underline{NT}^2
1. 3 1, 2, 3	1. 2 1, 2
2. 3 1, 2, 4	2. 2 1, 3
3. 3 1, 5, 6	2. 2 1, 3

No segment reference is necessary since there will only be one segment.

\underline{LT}^1	\underline{LT}^2
1. A + 2 1 2	1. A + 2 2 1
2. A - 2 1 3	2. D - 2 2 1
3. B - 2 2 3	3. G - 2 1 2
4. C - 2 3 2	
5. D - 2 1 3	
6. E - 2 3 2	
\underline{TT}^1	\underline{TT}^2
1. V	1. C 2 (points to CT)
2. V	2. V
3. V	

Note: It may be convenient to maintain the rules in this format; originally, however, it is not required for this process.

These tables are needed only temporarily since resolution will result in the appendage of these tables to the permanent tables. In order to unify one of these sets, we must unify the first literal of \underline{LT}^1 or \underline{LT}^2 (A) with the rightmost literal of the parent node #2, which is literal #1 of $\underline{LT}(\bar{A})$.

B(4) Mechanism for Rule Resolution & Example

The following steps are done:

In the rules NT, delete the first literal pointer (this is the resolving literal which is dropped) and add to the remaining pointers $M = (\text{number of literals in literal table } 1)$.

Example (Refers to tables \underline{NT}^1 , \underline{LT}^1 , and \underline{TT}^1)

$M = 3 - 1 = 2$ \underline{NT}^1 becomes

1. 2 4, 5
2. 2 4, 6
3. 2 7, 8

In the rule LT add to the term pointers $N = \#$ of terms presently in the term table of the parent.

Example: $N = 2$ \underline{LT}^1 becomes

1. A + 2 3 4
2. A - 2 3 5
3. B - 2 4 5
4. C - 2 5 4
5. D - 2 3 5
6. E - 2 5 4

Create a new segment of the term table by copying over the TT segment of the parent and appending the rules TT.

Example: Create segment IV of the TT.

IV 1. V
 2. C 2
 3. V
 4. V
 5. V

Unify the terms of the first literal of the rules LT with the terms of the rightmost literal of the parent node as follows:

- If the parent's term is a variable, link that term to the rule's new term.
- If the parent's term is constant and the rule's term is variable, link the rule's term to the parent's term.
- If both terms are constant, set both constant pointers to same location in constant table.

Example: Unify terms of 1st literal of $LT^1 = 3, 4$ with terms of literal #1 in $LT = 1, 2$.

In TT segment 4
1 and 3 are both variable so link 1 to 3
2 is constant and 4 is variable so link 4 to 2

IV 1. V 3
 2. C 2
 3. V
 4. V 2
 5. V

Append the rules LT to permanent LT except for the first entry (conclusion).

For each entry in the rules NT, create a new entry in NT with the same literals as the parent node, with the last one framed, followed by the added literals. Set the term segment pointer equal to the new segment.

NT (continued)

4. 3 1 4, 5 IV
5. 3 1 4, 6 IV
6. 3 1 7, 8 IV

LT (continued)

4. A - 2 3 5
5. B - 2 4 5
6. C - 2 5 4
7. D - 2 3 5
8. E - 2 5 4

After applying the same procedure with the second unification class, two additional nodes are created. The final structures describing all 8 nodes are as follows:

<u>NT</u>			
<u>ID#</u>	<u># of Lit.</u>	<u>Pointers to LT</u>	<u>Pointer to TT seg.</u>
1	3	1, <u>2</u> , 3	I
2	1	1	II
3	2	1, <u>2</u>	III
4	3	<u>1</u> , 4, 5	IV
5	3	<u>1</u> , 4, 6	IV
6	3	<u>1</u> , 7, 8	IV
7	2	<u>1</u> , 9	V
8	2	<u>1</u> , 10	V

<u>LT</u>				
<u>ID#</u>	<u>Relation</u>	<u>Sign</u>	<u>Arity</u>	<u>Pointers to TT</u>
1	A	-	2	1 2
2	B	-	2	3 1
3	C	-	2	3 1
4	A	-	2	3 5
5	B	-	2	4 5
6	C	-	2	5 4
7	D	-	2	3 5
8	E	-	2	5 4
9	D	-	2	4 3
10	G	-	2	3 4

<u>TT</u>			
<u>Segment</u>	<u>Term ID</u>	<u>Term Type</u>	<u>Subst Value</u>
I	1.	V	
	2.	V	
	3.	C	1
II	1.	V	
	2.	C	2
III	1.	V	4
	2.	V	
	3.	C	1
	4.	C	3
IV	1.	V	3
	2.	C	2
	3.	V	
	4.	V	2
	5.	V	

V	1.	V	4
	2.	C	2
	3.	C	2
	4.	V	

This method has the benefit of automatically separating variables, i.e., a variable in a resolving rule, which is not being unified, will never match a variable in the parent node.

B(5) Possible Optimizations

- (1) Instead of adding all terms from the rules TT, just add the new terms and change the old references. Furthermore, if no new terms are added, no new TT segment need be created.
- (2) As resolution proceeds and framed literals are dropped, terms are lost. However, these terms are still kept in the term table.

For example, on the previous page, look at node 3. Literal #2 may be contracted. Once this happens, node 3 has only one literal, number 1 which references two terms in segment III - 1 and 2. In segment III, term 1 is linked to term 4. However, there is no reference to term #3 in segment III.

Terms which are not referenced could be deleted from the segment. This would, however, require explicit ID numbers in the term table.

- (3) After repeated resolutions, it is possible to create a number of levels of pointers, i.e., term 1 is linked to term 4 which is linked to term 7, etc. This situation can be improved by updating pointers, so that when term 4 is linked to term 7, all other terms which were previously linked to term 4 (e.g., term 1) are linked to term 7 as well.

APPENDIX F

REFERENCES

REFERENCES

1. Symbolic Logic and Mechanics Theorem Proving. Clary, C. L. and Lee, R. C. J., Academic Press, New York, New York, 1973.
2. Design Concept for an Augmented Relational Intelligence Analysis System (ARIAS). Auerbach Report 2022-TR-2 1973, Sable, J., et al.
3. The Complexity of Resolution Procedure for Theorem Proving in the Propositional Calculus. Zvi Calli TR-75-239, Cornell University, 1975.
4. TT-Representation - A Clause Representation for Parallel Search. Fishman, D. H. and Minker, J., TR 74A-u, University of Massachusetts, 1974.
5. A CM-SIGMOD International Conference on Management of Data. San Jose, California, May 1975. Re Relational Data Nets in Information Systems.
6. Resolution Refinements and Search Strategies. A comparative Study. Wilson, G. and Minker, J. University of Maryland, July 1975.
7. Artificial Intelligence. The Heuristic Programming Approach. Slage, J. R., McGraw-Hill Series in Systems Science, 1971.
8. Computational Logic: The Unification Computation. Robinson, J. A., MacLume Intelligene 6, Amer. Elsevier (1971).
9. Problem Solving Methods in Artificial Intelligence. Nilsson, N., McGraw-Hill, New York, New York, 1971.
10. A Problem-Oriented Search Procedure for Theorem-Proving. Fishman, D. H., Coins TR 75-C-4, University of Massachusetts, June 1975.
11. Some Special Purpose Resolution System. Kuehne, D., Machine Intelligence 7, Wiley (1972) pp. 117-128.
12. And-or Graphs, Theorem-Proving Graphs and Bi-directional Search. Kowalski, R., Machine Intelligence, Wiley (1972) pp. 167-194.

*MISSION
of
Rome Air Development Center*

RADC plans and conducts research, exploratory and advanced development programs in command, control, and communications (C³) activities, and in the C³ areas of information sciences and intelligence. The principal technical mission areas are communications, electromagnetic guidance and control, surveillance of ground and aerospace objects, intelligence data collection and handling, information system technology, ionospheric propagation, solid state sciences, microwave physics and electronic reliability, maintainability and compatibility.

